Evolution of Architectural Floor Plans

Robert W. J. Flack

Computer Science

Submitted in partial fulfillment
of the requirements for the degree of

Master of Science

Faculty of Computer Science, Brock University
St. Catharines, Ontario

Acknowledgements

I would like to thank the following individuals and organizations for their support in preparing this thesis:

- B. Ross for superb supervision
- B. Ombuki-Berman and S. Houghten for participating on my supervisory committee
- C. Fairchild for technical assistance
- NSERC for generous funding

Abstract

Layout planning is a process of sizing and placing rooms (e.g. in a house) while attempting to optimize various criteria. Often there are conflicting criteria such as construction cost, minimizing the distance between related activities, and meeting the area requirements for these activities. The process of layout planning has mostly been done by hand, with a handful of attempts to automate the process. This thesis explores some of these past attempts and describes several new techniques for automating the layout planning process using evolutionary computation. These techniques are inspired by the existing methods, while adding some of their own innovations. Additional experiments are done to test the possibility of allowing polygonal exteriors with rectilinear interior walls. Several multi-objective approaches are used to evaluate and compare fitness. The evolutionary representation and requirements specification used provide great flexibility in problem scope and depth and is worthy of considering in future layout and design attempts. The system outlined in this thesis is capable of evolving a variety of floor plans conforming to functional and geometric specifications. Many of the resulting plans look reasonable even when compared to a professional floor plan. Additionally polygonal and multi-floor buildings were also generated.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Floor Plan Design

As computing time is becoming cheaper and more available, computers are being used for an expanding variety of creative endeavours. Some of these recent endeavours aim to put creativity in the computers' hands. Computer algorithms are being written to do everything from redesigning circuits to creating artwork or even writing poetry. There have been an increasing number of successes in the attempts to generate creative design results from evolutionary processes.

Architecture is a complex amalgamation of science and art. There are functional requirements, cultural expectations and general guidelines to follow, but within these guidelines there are still limitless possibilities. Even though a house may meet building codes and social norms, Hirsch feels that there is no such thing as **the** perfect house, "The needs and desires of every client are so unique, so it follows that each should should necessarily be unique."[15] It's likely that no amount of standard measures can identify one house that will suit everyone. This makes the design of houses an interesting problem to assist with a computer algorithm.

This thesis is concerned with the automatic design of the floor plan of a house. There are very many levels of detail that go into designing a house. A floor plan needs to be functional in that it shapes the flow of traffic through the house and it also has measurable traits of quality, such as efficiency in being able to get to important rooms quickly or being able to light key rooms with natural light from large windows. There are also conventions that

different societies have developed and expect to find satisfied with respect to their floor plans and this is why it is an ideal target for automation. There are many objectives that can be measured in the automation of the design. In particular, a floor plan needs a certain number of rooms of certain types. One should be able to get to all of the rooms in the house without travelling through too many other rooms to get there. Many rooms need to be a certain size and general shape or ratio of width to length. Certain rooms should be closer together. Some rooms need windows for natural lighting during the day. The type of walls, awnings over the windows, various doors and light fixtures all contribute to the feel of a house, but they are fairly immeasurable in terms of correctness. Additionally such decorative decisions can be made and applied to any house floor plan. The most important aspect of a house is that its owner likes it. The opinion of the owner is however very subjective. Owners may not agree on what is important and it is not necessarily quantifiable. They simply have to see the house. That is what makes this a particularly difficult and in a way a creative problem.

This thesis will explore several methods and ideas with respect to automating the generation of floor plans using a computer system. There are many objectives which are often at odds with each other. For example, there is a minimum number of various room types but also a minimum size for each of these room types and an overall finite amount of space. This means sacrifices must often be made in either the size of the rooms or the number of such rooms. Additionally it is a very subjective problem and as such some solutions may be undesirable for purely aesthetic or other unmeasured or unmeasurable reasons. Genetic algorithms and genetic programming paradigms will be utilized as they are capable of exploring large search spaces in parallel and produce a set of answers rather than one single answer. Multi-objective strategies will be used to find a balance of the many objectives being optimized. By providing multiple answers a customer will get many options which satisfy the fitness constraints in different ways and combinations.

This thesis will investigate whether this strategy of evolution will be successful in creating designs that may have been designed by humans. A system that can create such designs has many possible applications. It can be used as a basis for ideas to be taken into consideration by architects and their clients. It can also be used to create dynamic environments for games. This goal is two-fold; for games, it can provide dynamic environments such that each time playing through is different, and it can create very large expansive environments that would have taken a long time to design by hand. Lastly,

it can be used for computer animations or movies, where generating large environments for background scenery would otherwise be very time consuming. In any case, it is an interesting and challenging evolutionary design problem for genetic algorithms and genetic programming which is only beginning to be studied.

1.2 Goals

1.2.1 Effectiveness of Evolutionary Computation Techniques

This thesis tests the effectiveness of several evolutionary computation techniques and multi-objective evaluation strategies in the context of a difficult and highly dimensional problem.

- A genetic algorithm strategy can be compared with a similar genetic programming strategy.
- Different multi-objective strategies can be compared.
- Diversity preservation strategies can be employed and tested for effectiveness in generating more unique solutions.

It is an interesting and challenging problem in evolutionary design. The goal is to gain insight into the relative effectiveness of these evolutionary computation technologies in the floor plan design problem.

1.2.2 Domain Application

This thesis also attempts to solve a domain problem in architecture which has not been thoroughly studied. This problem is the automatic generation of floor plans that satisfy the constraints and problems presented in architectural design. These difficult to measure or even define.

- An inhabitant must be able to live within the house comfortably with the rooms that they have.
- The rooms must be able to comfortably accommodate the furniture within, while still being able to navigate the rooms.

- It should be easy to navigate the house. Inhabitants shouldn't have to travel far to move around the house during their daily activities.
- All rooms in the house must be accessible.
- Common gathering rooms should be well lit with natural lighting during the day.

These goals are difficult to quantify or measure. Also it is more important to have solutions which are generally good in all of these attributes over ones which are optimal in one or two of these. A system which finds good solutions could be used as a design exploration tool for architects to use to find a basis for their plans. It is also a potential tool for game design systems where dynamically generated content would be preferred or manual content creation may be too time consuming.

1.3 Subsequent Sections

Subsequent sections are laid out as follows. Background information regarding evolutionary computation, genetic algorithms, genetic programming, and floor planning is found in Chapter 2. A literature review of previous work in similar problems is outlined in Chapter 3. The system designed for this thesis is described in detail in Chapter 4. Various evolutionary parameters and house requirements are tested on a basic floor plan in Chapter 5. Chapter 6 shows the capabilities and flexibility of the system in a series of more advanced problems. Chapter 7 provides a discussion of the advantages of this system and comparisons to previous work. Finally, Chapter 8 sums up the effectiveness of this system in meeting the outlined goals and notes potential future work that could be done to improve the system.

Chapter 2

Background

Evolutionary computation is a subset of the broader field of artificial intelligence. Artificial intelligence or AI does not have a single definition. Its definitions range from systems that think and act like humans to systems that think and act rationally[27]. In general it is concerned with "the scientific understanding of the mechanisms underlying thought and intelligent behavior and their embodiment in machines."[1] Machine learning is the branch of artificial intelligence in which systems improve their performance over time through some form of experience. Evolutionary computation is the branch of machine learning in which the concept of evolution through natural selection is applied.

2.1 Genetic Algorithms

A genetic algorithm[10][22] is one of the more popular evolutionary systems which models a population of individuals evolving through natural selection as made famous by Charles Darwin[7]. GA's simulate natural selection by modelling a population of individuals which reproduce sexually with individuals that have been stochastically selected to have been more fit. These are considered the individuals that would have survived and been selected for mating.

The individuals are represented by a fixed length chromosome. When designing a genetic algorithm the experimentor decides how this chromosome string (genotype) will be converted into a problem solution (phenotype). This in some ways restricts the set of possible solutions created by the system; the

representation and resulting chromosome length ultimately decide the size of the search space. Genetic algorithms are also not guaranteed to be optimal as virtually all of the steps involve random elements. This is viewed both positively and negatively. In one regard, the system is not guaranteed that it will ever find the best solution. On the other hand, this stochastic nature means that it can quickly hone in on good solutions in the population as it does not need to search everything.

2.1.1 GA Algorithm

The GA algorithm in Figure 2.1 begins by creating a list of randomly generated individuals or chromosomes, which represent solutions to the problem. Chromosomes are typically represented as strings of characters which can be transformed into problem solutions. These solutions are evaluated according to some defined fitness function. There is a some termination criteria that is checked at this point to see if the GA should continue running. Usually this is a number of generations, a specific target fitness, or the lack of change in fitness for some number of generations. If the termination criteria has not been met, a new population is created. This is done through the use of genetic operators such as crossover and mutation. These stochastically select more fit chromosomes from the current population, perform some alterations or combinations to them, and insert them into a new population for the next generation. Since more fit chromosomes are favored in the selection process, the population tends to have more and more fit chromosomes with each generation. Eventually this process is said to converge. That is, the fitness of the chromosomes in the new population are no better than those in the current population. There are many causes for this. Most often it is due to the population being clustered around similar or even identical good solutions.

2.1.2 GA System Design

There are several elements of a GA system which can be tailored to suit the problem at hand.

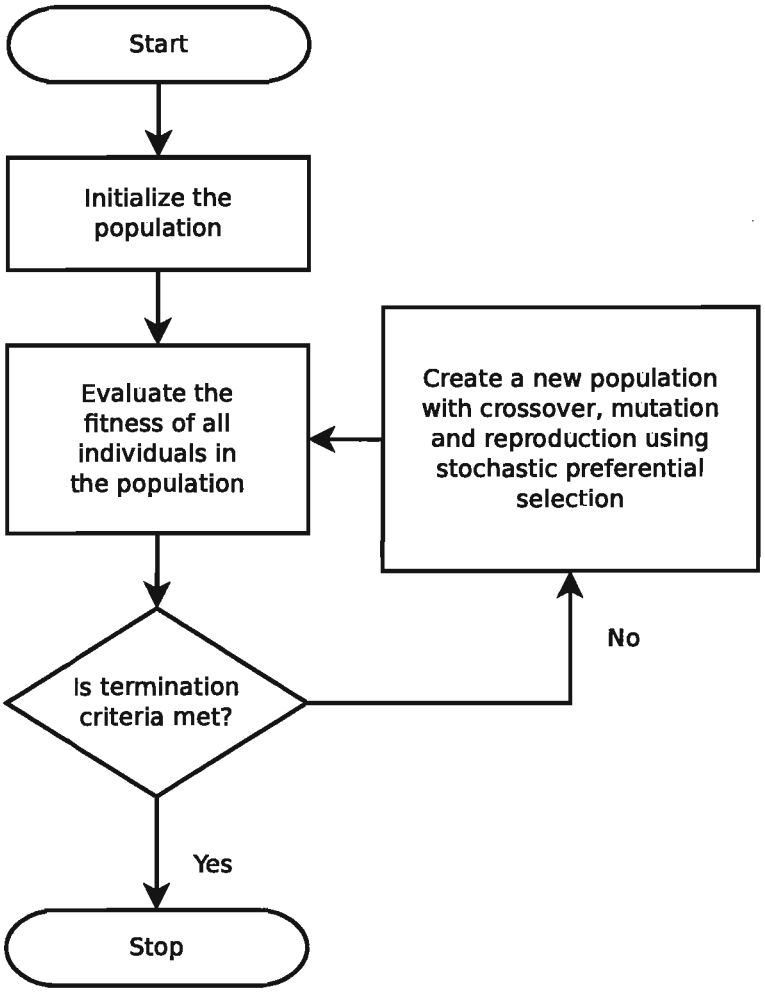


Figure 2.1: GA Algorithm

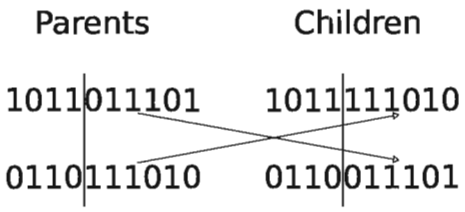


Figure 2.2: Vanilla GA crossover

Chromosome representation

The chromosome in a genetic algorithm represents a complete solution to the problem at hand, even though it may not necessarily be a good solution or not even a valid solution. While the representation will probably allow for bad or invalid solutions, all that is necessary is that it is capable of describing a good solution to the problem. Ideally it should be easy to randomly generate chromosomes for the beginning population. A vanilla GA describes a chromosome as a string of characters (usually 0 and 1), but in practice a chromosome can be any data structure.

Often a chromosome does not represent a direct solution to the problem. It must be transformed through some algorithm into a problem solution. This is akin to DNA in nature in that it does not represent the individual, but instead guides the development of an individual. In a genetic algorithm this transformation process should ideally be deterministic, otherwise the fitness of the resulting solution may not be representative of what is typically produced from that chromosome.

Crossover

The crossover operation or sexual recombination is an operation by which two parents are combined to create similar "child" chromosomes ideally incorporating the best traits of each parent. There are a variety of established crossovers for systems which use strings of characters as their chromosomes. The vanilla crossover for GA's is called a one-point crossover depicted in Figure 2.2. A single position is chosen within the chromosome string and the data after this position within each of the parents is swapped to create the two children. The resulting chromosomes are added to the population for the next generation. This is the most important reproduction operator

in GA's as it forms the basis for evolution. Traits from two individuals are combined to form new individuals. It is also the only reproduction operator necessary for evolution in a GA.

Mutation

While crossover is sufficient for many problems, it is often necessary to introduce mutation into a GA. This also fits into the analogy of Darwinian evolution; often children will have minor mutations giving them differences in traits from their parents. In a GA, mutation is typically performed on a small percentage of individuals and makes slight alterations to each of them so as to help the population explore the current solution area. In a vanilla GA with strings of characters as the chromosomes these slight alterations may just be changing some of the characters at random.

Fitness Proportional Selection

There are two things necessary in order for the population to evolve to a generally better population. A measure of what is better and a means by which better traits can be propagated to future generations. The measure of what is better is covered in the next section. The means of propogating the traits of better individuals is in the fitness proportional selection. The idea is to give better individuals a greater chance of being selected to propagate their genetic material. One standard mechanism for doing this is called tournament selection. During tournament selection n individuals are selected at random (where parameter n is the size of the tournament), and the best of these individuals is chosen for reproduction. Higher n values mean higher selective pressure. This algorithm is used for each selected parent for a crossover or mutation operation.

Fitness Function

A fitness function must be defined for the problem at hand. This function takes a chromosome, converts it into a problem solution if necessary, and provides a numerical value of how good of a solution it is. In a multi-objective genetic algorithm there are several numerical values produced by the fitness function. The accuracy of the fitness function in measuring partial success will have a dramatic effect on the genetic algorithms ability to evolve good

solutions, as it must know which solutions are "closer" to being good in order to breed those more often.

2.1.3 GA Parameters and Settings

In addition to the design elements there are many parameters and settings that can be adjusted to fine tune the search. This section describes parameters which have been applied to the runs in this thesis. Detailed explanations of these parameters and how they affect the results of a search are described in [22], [10], [16] and [27].

Population Size refers to the number of chromosomes in the population which is maintained from generation to generation. As each chromosome corresponds to one solution this is also the number of solutions maintained at each generation.

Probability of Crossover is the probability of using crossover when generating individuals for the next generation of the genetic algorithm. In the context of this thesis, this is used as the exact proportion of individuals generated with crossover. During crossover two parent chromosomes are selected and two new chromosomes are produced.

Probability of Mutation is the probability of using mutation when generating individuals for the next generation of the genetic algorithm. In the context of this thesis, as before, this is used as the exact proportion of individuals created using mutation. During mutation, one chromosome is selected, and a single chromosome is produced by making a minor random alteration to it.

Tournament Size is the size of the tournament used for the tournament selection in the GA. The tournament size can be set to any positive integer. If set to 1, there will be no selective pressure towards more fit individuals making the genetic algorithm essentially perform a random search. If set to any value greater than one, the GA will have fitness proportional selection in that individuals which are more fit will be more likely to be selected.

Diversity Preservation Factor is the rank value added to an identical individual in the population. Genetic algorithms sometimes have a

tendency to converge into a state where many individuals are identical. One way to help prevent this is by penalizing the fitness score of duplicate individuals. In the context of this thesis lower fitness scores are better, so an identical individual having it's fitness multiplied by this factor will be less valuable.

2.2 Genetic Programming

Genetic programming is an extension of genetic algorithms. Rather than using fixed-length strings of characters as the chromosomes or individuals a more complex hierarchical dynamically sized tree structure is used[16][25]. Typically these structures represent a computer program in a tree structure.

Cellular encoding is a technique used in developmental genetic programming whereby these tree structures are interpreted as instructions or operations performed on some simple intial structure.[25] The quality of the finished structure is taken to be the fitness of the program. This thesis makes use of a developmental genetic programming paradigm in the genetic programming form of the system.

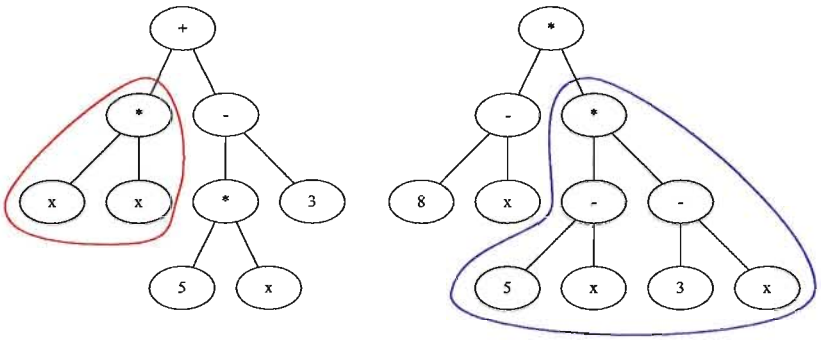
The GP algorithm is identical to the GA algorithm (See section 2.1.1), the difference is that since the chromosomes are hierarchical tree structures of variable sizes they cannot make use of crossovers designed for fixed-length strings. The standard GP crossover function is described in Section 2.2.1.

2.2.1 GP System Design

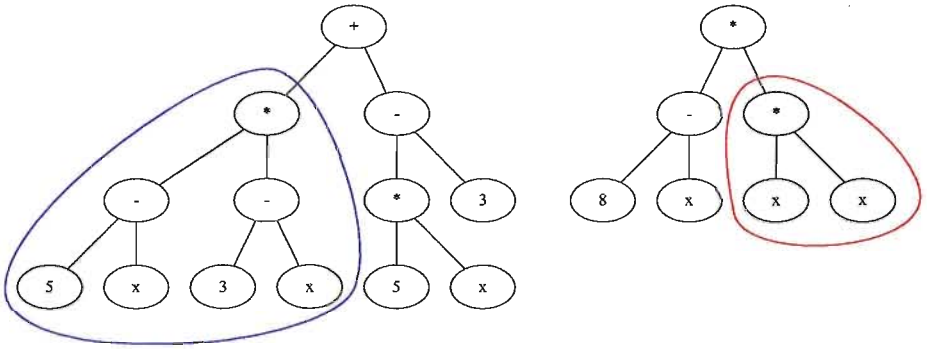
GP Chromosome Representation

Unlike genetic algorithms, the chromosome in use in genetic programming is a variable size tree structure. It is also referred to as a program or individual. This tree structure consists of non-terminal nodes (those which have children, also known as branch nodes) and terminal nodes (those which have no children, also known as leaf nodes). These correspond respectively to functions that take parameters and terminals which have no parameters. For example, in an arithmetic expression an addition function would be a non-terminal node that has two children whose values it adds up whereas the variable x or value 5 are examples of terminal nodes.

The function and terminal sets make up the building blocks the genetic programming system can utilize in solving the problem. In traditional GP's



(a) Parent chromosomes and selected subtrees



(b) Resulting children chromosomes from swapping selected subtrees

Figure 2.3: GP crossover

and in this thesis all of these functions and terminals are capable of operating on any node in the tree. There are no types or grammars restricting the possible combinations. All functions must ensure closure meaning that it must be able to produce a result for every possible set of inputs. For example a division function in a GP needs to be protected for division by zero if zero is a possible input.

As the function and terminal sets are the building blocks available for solving the problem, they should be capable of solving the problem in a direct way.

Crossover

The crossover operation in a genetic program works by selecting a node in the trees of the two selected parent chromosomes or programs. There is often a configurable ratio which determines how often an interior (non-leaf) node is selected and how often a terminal (leaf) node is selected in the tree. Once a node has been selected in each parent chromosome, two new chromosomes are produced by swapping the subtrees contained under those selected nodes in the parents. Figure 2.3 shows an example of this crossover operation. If either of the resulting trees exceed the maximum depth or number of nodes prescribed by the GP settings then this operation is considered a failure and the GP must retry it.

Mutation

The standard mutation operation in a GP selects a random node in an individual and replaces the entire subtree at that node with a new randomly created subtree. If the individual's overall maximum depth or number of nodes is now greater than the prescibed maximums in the GP settings then this operation is considered a failure and the system must retry it.

2.2.2 GP Parameters and Settings

The GP shares all of the parameters and settings of the GA and has a few of its own. The shared parameters can be seen in section 2.1.3. In addition to the GA parameters, there are parameters concerning the shape and size of the trees constructed by the GP.

Minimum Initial Tree Depth refers to the minimum tree depth permitted in the initial population creation phase.

Maximum Initial Tree Depth refers to the maximum tree depth permitted in the initial population creation phase.

Maximum Tree Depth restricts the maximum tree depth of any individual throughout the entire GP run.

Maximum Number of Nodes restricts the maximum number of nodes in any individual throughout the entire GP run.

Probability that Crossover Point is a Branch is used during the crossover operation. It determines the probability that the randomly selected node during crossover is a branch or non-terminal node.

Maximum Regenerative Depth for Mutation refers to the maximum depth tree created to replace a node during the standard mutation operation.

Maximum Number of Retries restricts the number of times the GP system can attempt to successfully breed an individual. During the standard crossover and mutation operations, there are no guarantees that the resulting trees will conform to the prescribed maximum number of nodes and tree depths. In the event that the created trees exceed these limits, the operation must be attempted again. If the operation fails to be successful after this many attempts the source individual is copied into the new population.

2.3 Multi-objective Evaluation

Often when solving a problem there are many criteria of measurement. Furthermore, these criteria often conflict with each other such that optimizing any one criteria would result in sacrificing the others. The floor plan layout problem presented in this thesis is no exception.

As this is a common problem, there are a variety of ideas by which one can attempt to strike a balance between competing objectives. These ideas and their benefits and weaknesses will be examined in this section.

2.3.1 Weighted Sum

The most common and simplest means of optimizing more than one objective is by using a weighted sum calculation. The overall fitness of an individual is the summation of the fitnesses of each of its objectives multiplied by weights for the objectives. The following formula shows how the final fitness score is derived for k objectives with weights $w_1, w_2, ..., w_k$.

$$fitness = w_1 * f_1 + w_2 * f_2 + ... + w_k * f_k$$

The advantage of this method is that it is simple to implement. The weights can be adjusted to prioritize important objectives, and the resulting

overall fitness values can be compared directly as they all have the same scale. The disadvantage is that the specific weighting on the objectives can have a great impact on the algorithm's success due to a "user bias". If the scale of one objective is too much larger than the rest than this objective is likely to be the only one that is optimized.

2.3.2 Pareto Ranking

Pareto ranking is often used for problems in which there are many objectives that are difficult to weigh against each other. It has been used to find good solutions in many multi-objective problem areas[10]. A solution is said to dominate another solution if the fitness in each of its objectives is at least as good as the other solution, and has a better score in at least one of the objectives as described by the following relation.

$$A \text{ dominates } B \Rightarrow (\forall_{obj}\, f_{obj}(A) \leq f_{obj}(B)) \wedge (\exists_{obj}\, f_{obj}A < f_{obj}B)$$

A solution is said to be Pareto optimal if it is not dominated by any other solution in the population. All Pareto optimal solutions are given a pareto rank of 0. The remaining non-optimal solutions are then examined, and the Pareto optimal solutions among those are given rank 1, and the process is repeated until all solutions have been ranked. These ranks are used as a single fitness value by which to compare individuals.

Figure 2.4 shows an example of this ranking. In Figure 2.4(a) the raw fitness values in all three objectives of the five individuals are shown. Using these objective values one can determine which individual dominates which to produce the matrix in Figure 2.4(b) where each row shows which individuals the individual of that row dominates. From the dominance matrix, one can see that solution A and C are undominated (as columns A and C have no X's). These Pareto optimal solutions become rank 0 and are no longer considered. Solution D is now undominated becoming rank 1. Solution B, no longer dominated by D is now optimal becoming rank 2. Lastly solution E is given rank 3.

The advantage of Pareto ranking is that any potentially ideal solutions are Pareto optimal and given rank 0. There are no sacrifices made with respect to one objective over another. However, this is also a disadvantage. A solution that is optimal in one objective but does not satisfy the other

Ind	Obj 1	Obj 2	Obj 3
A	3	1	2
B	5	3	2
C	6	2	1
D	5	2	2
E	7	4	3

(a) Population fitness values

	A	B	C	D	E
A		X		X	X
B					X
C					X
D		X			X
E					

(b) Dominance matrix

Ind	Rank
A	0
B	2
C	0
D	1
E	3

(c) Pareto ranks

Figure 2.4: Example of Pareto ranking

objectives at all is still Pareto optimal. This is almost guaranteed to occur as objectives are often at odds with each other making it easy to optimize one without consideration for the other. The main downfall, which is especially true with many conflicting objectives, is that populations can easily become composed almost entirely of nondominated solutions. In this case, a genetic algorithm degrades into a random search as there is no selective pressure since no solution can be identified as better than any other.

2.3.3 Sum of Ranks

Sum of ranks is a less commonly used means of ranking a population of individuals with many objectives. It has been examined as a proposed solution to the problems with Pareto ranking on highly dimensional problems[6]. The idea is to rank each objective separately within the population. The sum of these ranks is used to provide an overall rank for the individual. The sum of dominance ranks may also be used in the event of many similar fitness values. The dominance rank is the number of individuals which have a better value. This thesis uses dominance ranks in the standard ranked sum.

Figure 2.5 shows an example of this. Figure 2.5(a) shows the raw fitness values for each of the individuals in all objectives. Figure 2.5(b) shows the dominance rank within the population of each objective value for each individual. These ranks are added together to get the overall rank of each individual in the last column.

The advantage of using the ranked sum fitness evaluation is that there is a greater diversity in the resulting ranks over that produced from Pareto ranking. Having diversity in the ranks provides greater selective pressure towards solutions that are hopefully better overall. The result is that the algorithm can converge towards a solution that is good in all fitness objectives in cases where Pareto ranking would get stuck with an entire population of Pareto optimal solutions which are not necessarily good in all objectives.

Normalized ranks may be used to lessen the weight of highly similar objectives. Figure 2.5(c) shows an example of this. The rank of an individual is the number of ranks less than it. When computing the overall rank, each objective rank is divided by the number of ranks for that objective. In the example figure, objective 3 is not weighted as heavily in the overall rank as there are many individuals with the same fitness in this objective score. This favours individuals which excel in objectives that are less common to excel in.

Ind	Obj 1	Obj 2	Obj 3
A	3	1	2
B	5	3	2
C	6	2	1
D	5	2	2
E	7	4	3

(a) Population fitness values

Ind	Obj 1	Obj 2	Obj 3	Rank
A	0	0	1	1
B	1	3	1	5
C	3	1	0	4
D	1	1	1	3
E	4	4	4	16

(b) Ranked Objectives

Ind	Obj 1	Obj 2	Obj 3	Rank
A	0	0	1	0.500
B	1	2	1	1.500
C	2	1	0	1.000
D	1	1	1	1.166
E	3	3	2	3
# Ranks	3	3	2	

(c) Normalized Sum of Ranks

Figure 2.5: Example of Sum of Ranks

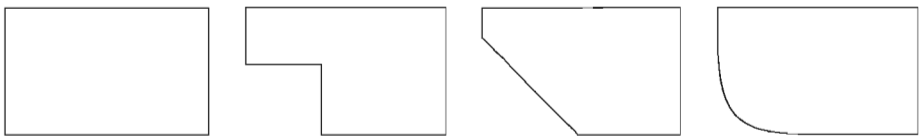
2.4 Floor planning

Designing a house for someone can be very tasking. While it may be easy enough to create something that is livable, creating a home that someone will want to invest a life in is another task altogether. As mentioned before, there is no such thing as the perfect house for everyone. The best that can be hoped for from an automated system is to give a variety of houses which meet general requirements, and hope that one or more of them may serve as inspiration for a client's dream house.

The requirements for a house are not all too explicit. While it is required that you be able to escape from the bedrooms in the event of a fire, there is no formal requirement that people shouldn't have to walk through the bedroom to get to the kitchen. There is no formal requirement that it should be easy to find the bathroom. These implicit requirements are derived from a combination of western culture and simple usability guidelines. They have been generally observed in most modern houses, and the ones that do not observe them usually give one that feeling as though something is out of place.

There are spatial requirements for a house. An implicit one is the overall footprint, the house must be built to fit within the space allotted for it on a lot. As a result of this room shapes and sizes must be designed accordingly to fit within the space. There are also various layout requirements, and functionality requirements. In an autonomous system, these requirements either have to be implicit in the representation such that they are always satisfied, or explicitly measured as the quality of the solution.

It should be noted that these goals often vary from one culture to another, and one time period to another. As this paper is concerned with the constraints in the modern western society, the results and some of the goals may not coincide with other cultures. Nevertheless, many of these requirements could be tailored to other cultures as they are provided as part of the problem specification. What makes the requirements of a house particularly complex is that not all of the goals are strictly required or impose specific constraints on the design. Many of the requirements are flexible, and it is these requirements that are often highly subjective. Nevertheless, if only the required constraints were satisfied the results would likely not be pleasing.



(a) Rectangular (b) Rectilinear (c) Polygonal (d) Curves

Figure 2.6: Room shapes

2.4.1 Spatial requirements

There are a variety of spatial requirements that must be considered in the design of a house. First and perhaps most importantly is the overall footprint of the house. The footprint will ultimately define the layout of the house, as all of the rooms within the house must fit within the overall footprint. The location of the front door can be considered part of the footprint, and will be for the purposes of this thesis. This is usually a central point within the house as it is quite often necessary to leave the house. Although quite often the shape of the footprint is assumed to be rectangular, in practice this is not always the case. The number of floors is an equally important factor, again defining the overall space and shape of the house.

Within the house the room shapes and sizes is an important factor to be observed. Room shapes may be rectangular, rectilinear, polygonal or curves (See Figure 2.6). The size of the rooms factor into the overall shape, and functionality of the house. Hirsch[15] identifies deciding the target sizes of various rooms in your house as one of the primary discussions an architect has with their clients. More than specific sizes there are also implicit social norms such as a kitchen being larger than a bathroom. It is not simply a matter of looks, it must have room for a fridge, a stove, and a sink, and allow one or two people to easily maneuver through it in order to cook meals. A bathroom on the other hand only houses one person at a time, and may only requires a toilet and a sink. These spacial requirements can easily be deduced from existing houses as they have been observed in the design of virtually all modern houses in the western society.

It can be observed that while people often specify exact sizes that they wish the rooms in their house to be, these are often only suggestions. Typically the size of the house limits the size of the rooms within it, and a floor plan is only unpleasing if rooms do not have space for the activities and items

they will host. While there may not be an explicit maximum size for any room, having too much space will also tend to look awkward. Furthermore, we expect similar ratios of magnitudes; for example, houses that have large bathrooms often have even larger bedrooms. Hirsch states, "Each space or room needs to be the correct and appropriate size for its function and 'feel.' That means it should not be too large or too small. Room design is not a case where bigger is better"[15].

2.4.2 Layout requirements

There are many requirements in terms of the layout of the house. Connectivity is a firm implicit requirement in a house design. Every room must be accessible, via some route. While a completely inaccessible room may have its uses as a safe room it is certainly not the norm to have such a room. Additionally it has been observed that a room for public use should not require traveling through a private room to get to it[21]. For example, it would not be desirable to have to walk through the bathroom to get to the living room.

Various path lengths throughout the layout will play an important role in the quality of the house. For example, the distance required to get to a bathroom from any social room would be an important one to minimize in general. Similarly, a dining room (or other room which can be used for eating) should be very close, if not adjacent to, the kitchen. Beyond this, many people discuss with their architects certain rooms that they may feel it important to have adjacent in the floor plan. For example, it may be important for the kitchen to overlook the playroom so that a single parent can cook dinner while keeping an eye on their children.

The number of adjacent rooms to a single room typically cannot exceed four, however even four adjacent rooms tends to be too crowded for social rooms. Having limited numbers of rooms connected to others while minimizing path lengths and being able to completely access the house make designing the layout a rather complex optimization problem.

2.4.3 Functionality requirements

A functional living space requires certain basic rooms for its inhabitants. At the very least, it should have a kitchen, a bathroom, a bedroom, and a living room. This is quite evident in the construction of modern apartment buildings which have exactly these rooms. When scaling up, there tends to

be a desirable ratio of bathrooms to bedrooms, or more directly to number of inhabitants. These ratios are not defined anywhere, but rather implicitly derived from the usage of a house. For example, quite often reducing the distance to the bathrooms necessitates the addition of an extra bathroom.

This is also often one of the key discussion points identified by Hirsch[15] when an architect talks to his or her clients. They discuss with the client and create a list of rooms the client would like to have asking key questions such as whether the house needs works spaces, private offices or project rooms. Some people may find it important for there to be a designated master bedroom with it's own bathroom large enough to have "his-and-hers" vanity and sink areas. This list of rooms is a key requirement that an architect builds with their client before setting off to design their home.

Chapter 3

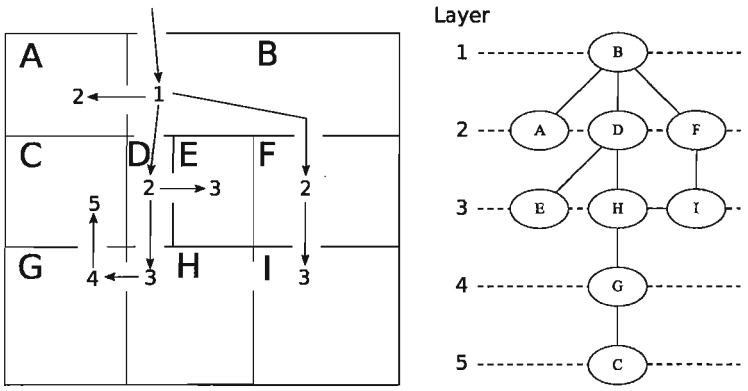
Literature Review of Floor Plan Design

3.1 Building Analysis

There are several ways to go about designing a building, although most strategies can be categorized by where they start in the design process. Some strategies begin with the exterior of a building, and it follows naturally that the interior must be designed to fit within the specified space. Other strategies construct an interior and it is this space that defines the shape of the exterior.

Hillier and Hanson[14] made some of the first observations in decoding social spaces. They observe that the connectivity graph between rooms is an essential component in analyzing the social use of space within a house. Furthermore, they define a "Justified Gamma Map" as follows. The depth of each room is determined by the number of steps needed to reach each room from outside. Then the rooms are placed in horizontal lines at a height relative to their depth. Lines are drawn between connected rooms. An example of a justified gamma map construction is shown in Figure 3.1. It is suggested that such a graph can decipher the underlying structure of buildings and show how they are similar or different. They also suggest that such a graph could be used in generating social spaces, however their focus was only on the analysis of social spaces.

The façade of a building is the first thing one sees, and it is the only thing that most people will see. The façade of a building gives the first



(a) Constructing gamma map breadth-first from entrance

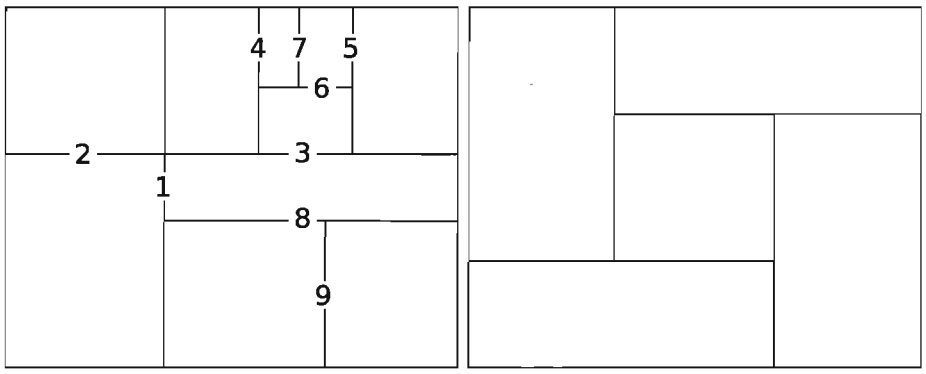
(b) Resulting justified gamma map

Figure 3.1: Justified Gamma Map

impression of the building, and as such there is no denying the importance in its design. Algorithms which construct building façades have been studied for many years. There have been methods which target efficient generation of façades and methods which attempt to design quality façades but perhaps require user input to facilitate their process.

Greuter *et al.*[11] describe a method in which "good looking" building facades are quickly generated. The algorithm begins at the roof of a building. First, a regular polygon is chosen and centered around one of the vertices of the current shape. The combined shape is then extruded downwards several floors. This process repeats until the ground floor is reached. The end result is that skyscraper style buildings are created very quickly. Müller *et al.*[24] describe a shape grammar by which multitudes of buildings can be evolved, resembling modern architecture through the use of well crafted generation rules. Through the intelligent combination of transformation rules in a shape grammar, a plot area is extruded and extended into some architectural form remeniscent of the intended era. Nevertheless, the end results are building exteriors and have not taken into account the ability to function as a social space. Furthermore, the rules necessary to generate these buildings need to be created by hand and require great attention to detail.

The problem of floor plan design is independent of that of façade creation with the exception that the overall shape must match that of the outside of the floor plan.



(a) Sliceable floor plan (b) A floor plan that is not sliceable

Figure 3.2: Sliceable and unsliceable floorplans

3.2 Floorplanning in VLSI

Very Large Scale Integration (VLSI) is a process of designing very large integrated circuits through the combination of thousands of transistors into a single chip which can be integrated into other designs. Floor planning is one of the first steps involved in VLSI. Module areas and interconnections are planned out usually with two goals in mind; reducing overall area and interconnection distances between related modules. Research has been done in automation of VLSI floor planning which can be compared to architectural floor plan design.

A sliceable floorplan is a floorplan that can be constructed by recusively subdividing (slicing) an initial single rectangular block[8]. Figure 3.2(a) shows an example of a floorplan that is sliceable. If one slices along the lines in the order of the numbers they will obtain that complete layout. Figure 3.2(b) on the other hand is not sliceable. Sliceable floorplans are convenient as they can be represented in the form of a tree structure of divisions. Also a number of NP-hard problems have polynomial time solutions when considering the set of sliceable floorplans[28].

Wong and Liu[33] use simulated annealing with a reverse polish notation representing the set of slices performed to divide the initial rectangle. By using simulated annealing they are able to optimize both overall area and interconnection distance at the same time. Wong and Liu also stress the importance of a using a representation that does not introduce too much

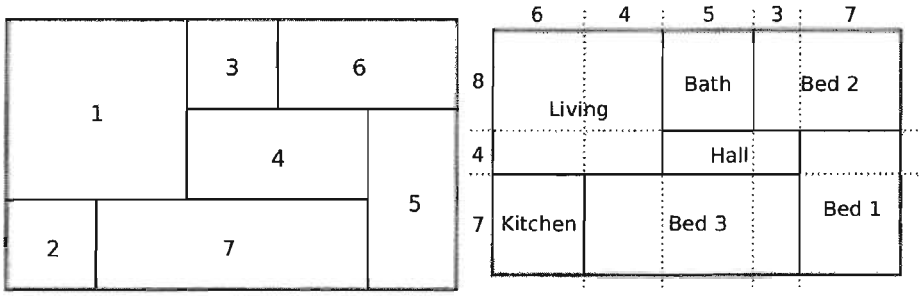
bias. Sutanthavibul *et al.*[30] demonstrate a linear programming solution which builds up a floor plan by adding a limited number of modules at a time. The number of constraints increases too drastically to plan the entire floorplan at once.

Many have successfully applied evolutionary computation to the design of VLSI floorplans. Lienig and Cohoon[18] use genetic algorithms with genetic operators incorporating expert knowledge to produce near-optimal designs for larger problems. Valenzuela and Wang[32] use a genetic algorithm with a specialized slicing tree encoding to produce area optimized floorplan solutions. Tang and Alvin[31] use a genetic algorithm with an ordered tree representation, introduced by Guo *et al.*[12], to reduce the search space size which consistently produces better results than the deterministic algorithm on which it is based. The literature suggests that EC algorithms have been successful in VLSI with careful chromosome design and modified crossover operators. However, the problem requirements and goals do not directly coincide with those in architectural floorplanning.

3.3 Non-Evolutionary Automated Design of Floor Plans

Hahn *et al.*[13] demonstrate a method of generating building interiors in realtime as they are explored. The generation is a procedural algorithm that follows the use of 11 simple rules to generate spaces reminiscent of office buildings consisting of hallways and rooms. Tutenel *et al.*[4] use a hierarchical rule-based placement algorithm to create furnished living spaces with a variety of features such as objects needing clearance around them or those that require a view of the TV. They use a heuristic value calculation to assign a best location for each feature at a time. Bruls *et al.*[5] propose a visual representation for trees called squarified treemaps which Marson and Musse[19] use to quickly generate balanced floor plans. Their method converts internal walls into hallways in order to ensure proper connectivity in the resulting house.

Mitchell *et al.*[23] present an optimization version for small instances of a layout problem. The problem is to arrange rectilinear rooms in such a way that maximizes the number of desired adjacencies. They enumerate all solutions which ensure those adjacencies, after which dimension and area



(a) Constructed dimensionless layout. (b) Dimensions adjusted according to room types.

Figure 3.3: Transformation of dimensionless representation[23]

constraints may be introduced. The rooms are layed out on a grid whose rows and columns may be resized to satisfy the spatial constraints as shown in Figure 3.3. Finding the ideal sizes becomes a problem of satisfying a set of variables for various formulas. Due to the exhaustive search being employed, they suggest an upper limit on the problem size of 8 rooms.

Martin[21] applies a multi-phase constructive algorithm with an emphasis on a fast approximate solution that can quickly construct a batch of houses. He uses a procedural algorithm tuned with various statistics concerning spatial constraints, and common room adjacencies. In the first phase, a graph of public rooms is constructed. The second phase adds on private rooms and "sticky" rooms such as linen closets. The third phase places the constructed graph within the floor space, and then the fourth phase expands the walls of the rooms using a pressure simulation. The construction process is a deterministic greedy algorithm that attempts to maximize adherence to four measured statistics on the house.

3.4 Evolutionary Automated Design of Floor Plans

Schnier and Gero[29] use a genetic program with a dynamic set of primitive functions in order to evolve designs similar to a given plan. As useful features are identified they are added to the function set similar to the creation of ADF's[17]. In order to value diversity individuals are only thrown out if they

do not match the plan in any way.

Doulgerakis[9] compares the problem of creating a social space to the Facilities Layout Problem (FLP). The FLP attempts to find the ideal allocation of activities addressing the connectivity issue. In its simplest form, it is an assignment problem of activities to existing spaces. The goal is to minimize distances between related activites. In its most complicated form, the FLP aims to construct the layout as well, addressing the spatial requirements. Doulgerakis considers the most complicated form, using a genetic programming algorithm to first construct the space using division of an initial rectangle. Activity assignment is accomplished by a procedural algorithm followed by the evaluation of of the space. He also considers polygonal spaces (See Figure 2.6), by allowing angled splits of rectangles.

Chapter 4

System Design

The house evolving system is comprised of a few house construction strategies, with an identical evaluation in order to fairly compare between them. Two variations on a genetic algorithm and a genetic programming solution will be considered. Common to two of these strategies is a procedural activity assignment algorithm inspired by Martin[21] and Doulgerakis[9].

4.1 System Operation

The basic operation of the system is outlined in Figure 4.1. The genetic algorithm or genetic program produces some chromosomes. Those chromosomes are converted to a physical floor plan as shown in Section 4.2. If the floor plan does not have room types, these are procedurally assigned using the algorithm described in Section 4.3. The resulting floor plan is evaluated using the fitness evaluation described in Section 4.5. The fitness scores are then given to the multi-objective evaluation scheme to generate single scores for use by the GA/GP to select individuals for reproduction in the next generation.

4.2 Chromosome Representation

There are two representations used for the creation of floor layout plans. One is a fixed-size representation that facilitates the use of a genetic algorithm in the evolution of a floor plan. The other is a tree structured divisive representation to be created by a genetic program.

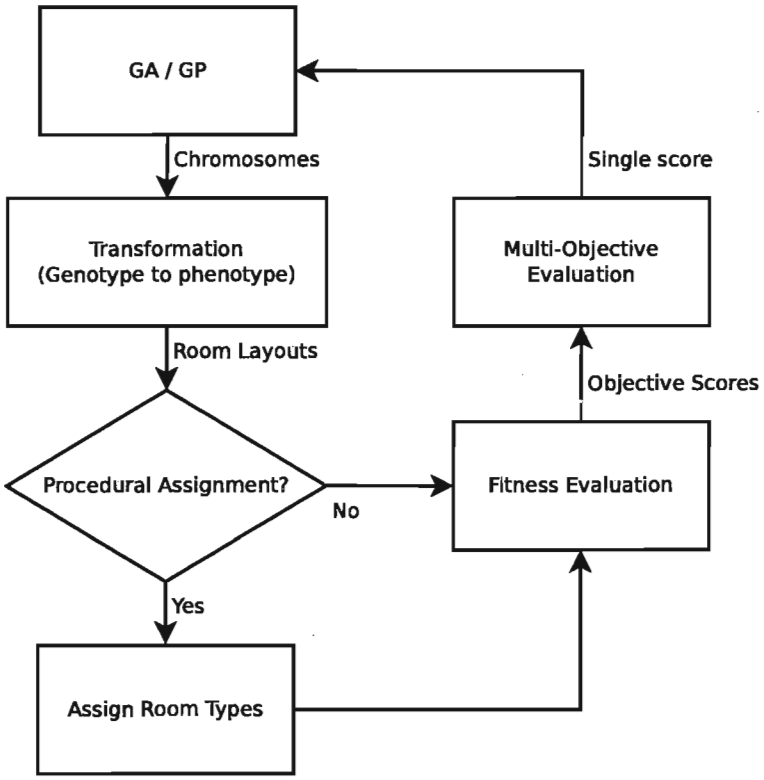


Figure 4.1: Flow of system operation

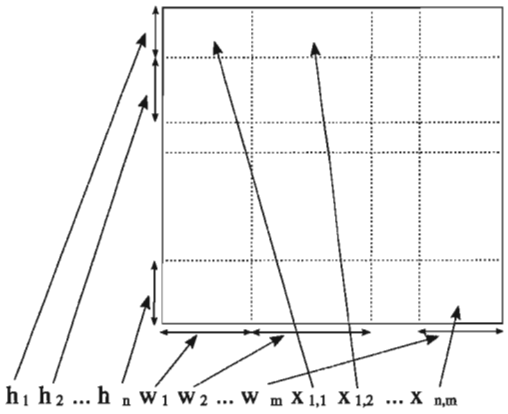


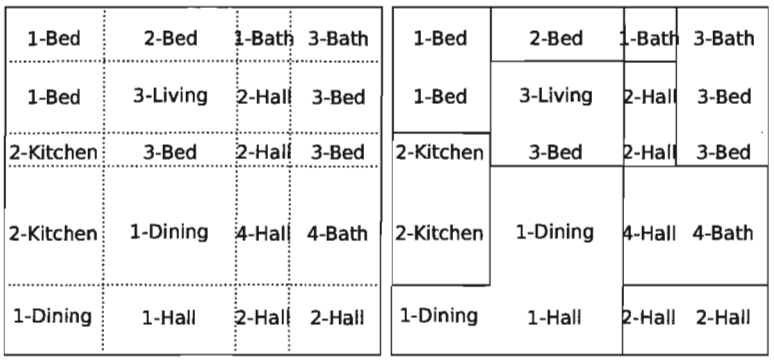
Figure 4.2: Chromosome mapping to phenotype

4.2.1 GA

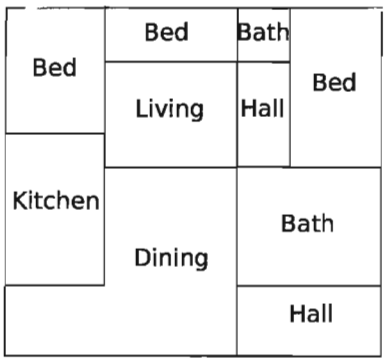
The system will use a grid similar to Mitchell *et al.*[23], whose size will be predetermined. If the house exterior is not rectangular the grid will be fit to the bounding rectangle of the exterior. The genotype representing a house configuration will contain the following. Figure 4.2 shows how the genotype maps to the floor layout phenotype. In this figure, $h_1, h_2, ..., h_n$ correspond to the heights of rows 1 through n. The values $w_1, w_2, ..., w_n$ correspond respectively to the widths of columns 1 through n in the grid. Lastly, $x_{i,j}$ corresponds to the room type and cell number of the cell in row i, column j. The width and height of the grid rows and columns can be resized and hence the representation is capable of describing virtually any rectilinear house given a grid of a high enough dimensionality.

- The size of each of the grid's rows and columns
- The room types of each location of the grid on all of the floors. If the procedural room assignment is being used than this type is either public, private, or no recommended room type.
- A cell number which determines which rooms this one will join together with. This allows for identical room types to not be later merged.

The transformation from genotype to phenotype proceeds by combining adjacent rooms of the same cell number into a single room, removing walls



(a) Grid from chromosome (b) Combination of cells



(c) Assigning type from a vote

Figure 4.3: Edge removal

between those grid cells as shown in Figure 4.3. If procedural assignment is being used the types will be public, private and no recommendation. The overall room type is given by a tally of the room types of all cells in each combined group. The final room type can be assigned using the procedural algorithm outlined in Section 4.3 if this method is being used.

Crossover between two individals is slightly different than a standard GA crossover. Given that the representation is in a table or grid, a rectangular selection is made by selecting two cell locations at random. Using this selection rectangle, the information from each parent is exchanged to create the children as shown in Figure 4.4

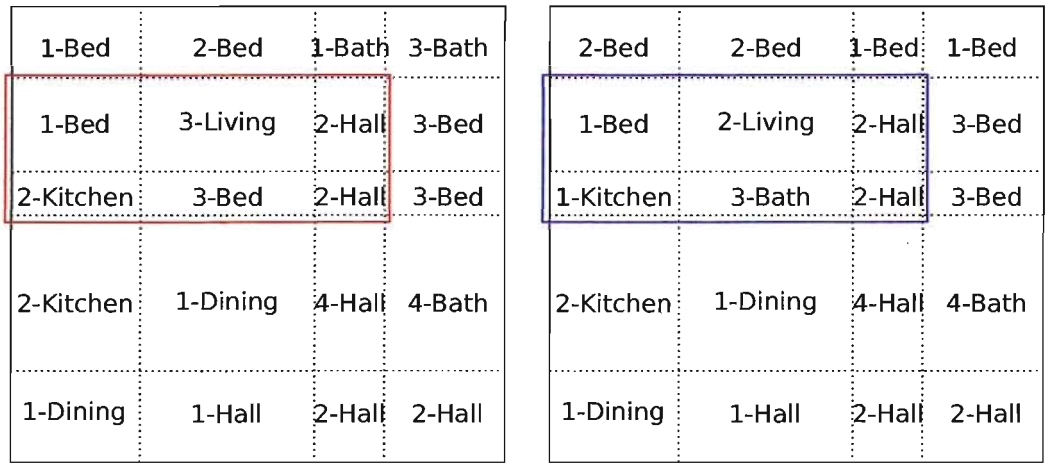
4.2.2 GP

The GP evolves tree-shaped individuals whose types specify their structure. Wong and Liu[33] use reverse polish representation to construct sliceable floor plans through repeated division of an initial rectangle. Doulgerakis[9] uses GP to create floor plans by the same recursive subdivision, except that the cuts may be angled.

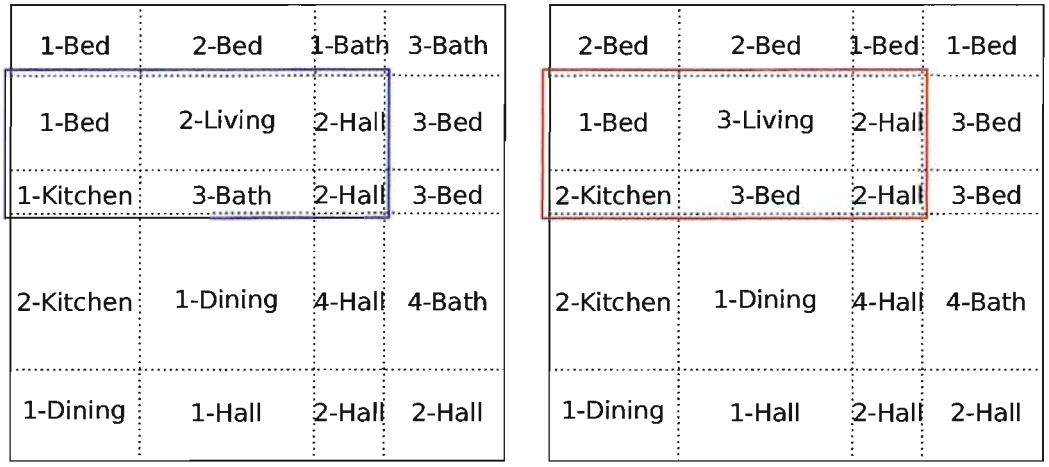
A developmental genetic program is used as one possible layout strategy. The GP embryo begins with the bounding box of the exterior house rectangle[1] and modifies it with the functions in the available function set listed in Table 4.1. The *Divide* functions will split the current block into 2 or 3 blocks and further processing can be done on these sub-blocks. The relative sizes of the new blocks are given by the arguments of the split operator. Alternately, the *Assign* function will automatically split on the longer of the two dimensions such that more regular rooms can be created. The terminals in the tree can be made to determine whether a room type is public or private.

Figure 4.5 shows an example of how a program tree is mapped to a floor layout. The process starts with a rectangle the size of the entire house. The first operation in the tree is a horizontal split, which splits the rectangle and uses the first subtree in the tree to construct the new smaller rectangle on the left. The second subtree is used to construct the other rectangle. The left rectangle is split vertically into three rectangles in the second step, and the first and the third of these rectangles are subsequently split horizontally in steps 3 and 4. Finally the large rectangle on the right is split vertically to make two regions.

[1]Or bounding box of a polygonal shape



(a) Parent chromosomes



(b) Children chromosomes

Figure 4.4: Example of crossover with GA house chromosomes

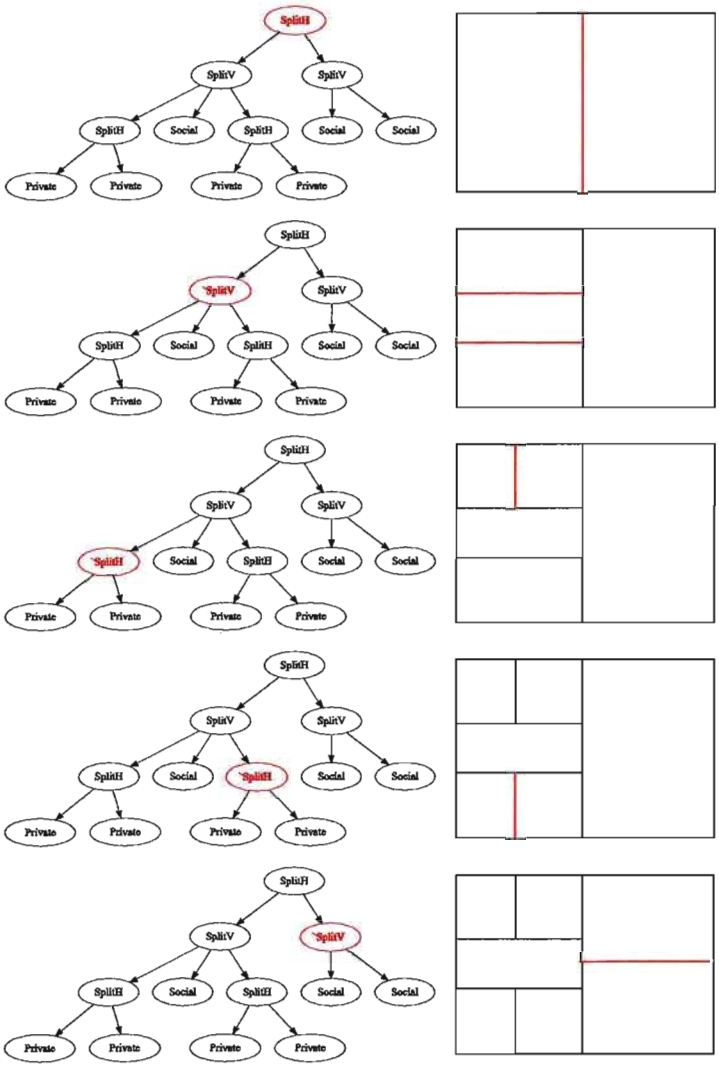


Figure 4.5: Developmental GP Layout Construction

Table 4.1: GP Function and Terminal Set

Function	# children	# arguments	Action
H-Divide	2 – 3	2 – 3	Divide the current rectangle horizontally into n regions with specified proportions.
V-Divide	2 – 3	2 – 3	Divide the current rectangle vertically into n regions with specified proportions.
A-Divide	2 – 3	2 – 3	Divide the current rectangle along its longest dimension (height or width) into n regions with specified proportions.
Assign	0	1	Assign the specified room type (or public/private designation) to the current rectangle.

Following the creation of the space, the procedural algorithm outlined in Section 4.3 can be used to assign the actual activity spaces.

4.3 Procedural Activity Assignment

Martin and Doulgerakis both use procedural algorithms to define their room types. Whereas Martin's algorithm begins with a graph of rooms defining their connectivity, the rooms are "size-less" and so this is not a consideration during assignment. Doulgerakis uses a procedural algorithm to assign types after the room positions and sizes have been fixed. This system uses a strategy more like that of Doulgerakis. The room positions are fixed and assignment takes place afterwards.

Martin[21] uses a procedural algorithm for assigning room activities after a rough diagram of the floor layout is known. He shows promising results from the application of this algorithm to fairly random graphs. The algorithm considers basic functionality requirements, as well as respecting the reachability requirement that a person should not have to travel through private rooms to get to public rooms. The sizes and ratios are worked out afterwards.

Doulgerakis[9] uses a similar procedural algorithm in his thesis to assign room responsibilities. The algorithm considers each possible room type by evaluating several characteristics. These characteristics include the minimum and maximum areas for the room types, the desired ratio, the adjacent rooms and their position on a justified gamma map. The algorithm uses a greedy algorithm to choose the room type that maximizes this evaluated criteria for each assignment and proceeds until there are no more rooms to assign.

The procedural assignment used for this thesis will borrow some ideas from both Martin and Doulgerakis. Each room will be assigned a type in turn. The process begins with the first room connected to the front door, which will be given a type that is acceptable to be connected to an entrance. The algorithm then examines all of the adjacent rooms and determines a list of types which each of the adjacent rooms can be. Each of the adjacent rooms are added to a queue to have their room types assigned in breadth first order.

To assign a room type when there are multiple possible types the procedural algorithm considers the following criteria, and chooses the "best" choice via a weighted evaluation of the following criteria. The exact penalties and

bonuses are detailed in Appendix A.

1. The room's size is compared to other rooms which have already been assigned to see if the size relations defined in the layout file still hold. If not, this room type is given a strong penalty.

2. If the room has too many twists[2], it will receive a strong recommendation to become a certain room type as specified by the requirements (typically a hallway).

3. If this type does not require a window, apply a penalty to rooms on the outer wall as they are best saved for window rooms.

4. If the chromosome suggests that the room should be a private / public type and the type is the opposite then apply a penalty.

5. If the room does not meet the minimum area or width or exceeds the maximum width or area then a penalty is applied.

6. If there is a desired width or height add a penalty for how far away from the desired values the room size is.

7. If the room has a ratio outside of the acceptible minimum and maximum ratio a penalty is applied.

8. If this room puts an adjacent room over the maximum number of adjacent rooms of this type then penalize it.

9. Check if this room helps meet the required number of rooms in the house. If so, give it a small bonus. The reason for a small bonus is that there will probably be many opportunities to place this room and it would not do well to place it as early as possible all the time.

10. Similarly, if this room pushes the house over the maximum number of this room type then penalize it.

The algorithm proceeds to assign room types unless it gets into a situation in which there are no possible types to assign. In such a case it backtracks and moves on to the next most valuable of the previous room assignment.

[2]Twists refers to the number of overlapping rectangles it takes to completely fill in the room.

4.4 Requirements Description

The requirements of a house are given for each problem. Since there is no inherent meaning in saying that a room is of a certain type, this meaning is imbued by the requirements given with the problem. The possible requirements for each type of room are described in this section.

Type refers to whether the room is a public or private room. This determines whether or not people normally travel through this room to get to other rooms.

Amount refers to the recommended minimum and/or maximum number of each type of room.

Twists refers to the minimum or maximum number of twists in the specified room type.

Width refers to the minimum width across any part of the room. It can have a recommended minimum or maximum size.

Area refers to the minimum, maximum or recommended area in the room.

Ratio refers to the maximum ratio of length to depth of any section of the room. It can have a recommended minimum and maximum value.

Windows refers to whether the room should be on an external wall so that it may have windows to let light in.

Access refers to the rooms which people in this room should be able to access in a small number of steps. The nearest access means that occupants only need to access any of the rooms of this type so the nearest one will do (for example a bathroom where any one will do).

Bigger refers to rooms which this room should be bigger than. This is used to define size relations where it makes more sense than defining absolute sizes.

4.5 Fitness Evaluation

4.5.1 Calculations

The calculation of an individual's fitness falls into several categories relating to its adherence to the specifications in the requirements (See section 4.4).

Functional measures the building's adherence to the living requirements. It is the sum of the number of missing rooms (room types where the minimum number has not yet been met) and the number of rooms in excess of the maximum numbers allowed.

Geometric measures the buildings closeness to idealized geometric measurements. The geometric score is the summation of several values. For each room where the area is greater or less than the maximum or minimum allowed area the difference in area is added to the geometric score. For each room where the minimum width across any rectangular region is less than the minimum or greater than the maximum required width, the squared difference is added to the geometric score. For each room, there is a list of rooms that the examined room should be bigger than. If it is smaller (in terms of area) than any of those room types, then for each one the amount it is smaller by is added to geometric score.

Connectivity measures how well the building satisfies certain proximities as specified by the requirements. A maximum distance value is set to be the number of rooms in the building as this is the limit on the furthest distance one would have to travel to get anywhere. If a room type has adjacency requirements but does not exist, the maximum distance score is given such that a house without certain room types is not rewarded for it. For all types that exist the geometric scores are scaled by the the inverse of the number of rooms of that type. This way a house is not penalized for having more rooms of a certain type, rather the average of the connectivity scores is computed. For each requirement to access the nearest of a room type from all other rooms, the distance to the nearest of each room type is averaged for all rooms and added to the score. For each requirement to access a room type from some type, the distance to the nearest room is calculated and added to the score.

Reachable is a measurement of how shallow and wide the graph of the building is. It measures the average number of rooms one must travel through from the entrance to reach any room in the building. It is desirable to keep this number low so that the building is not overly complicated and can be easily and efficiently traversed. If it is not possible to reach a certain room a value of 30 is added for that room.

Ratio measures how close the rooms in the building are to their recommended ratio requirements. This ensures that long and skinny rooms will be penalized when this is not desired. Ratio is measured in terms of the average of the larger $\frac{width}{length}$ measurement for each rectangular region in the room. For each room with a ratio below the minimum or above the maximum allowed, the difference in ratio is added to this objective as a penalty in score.

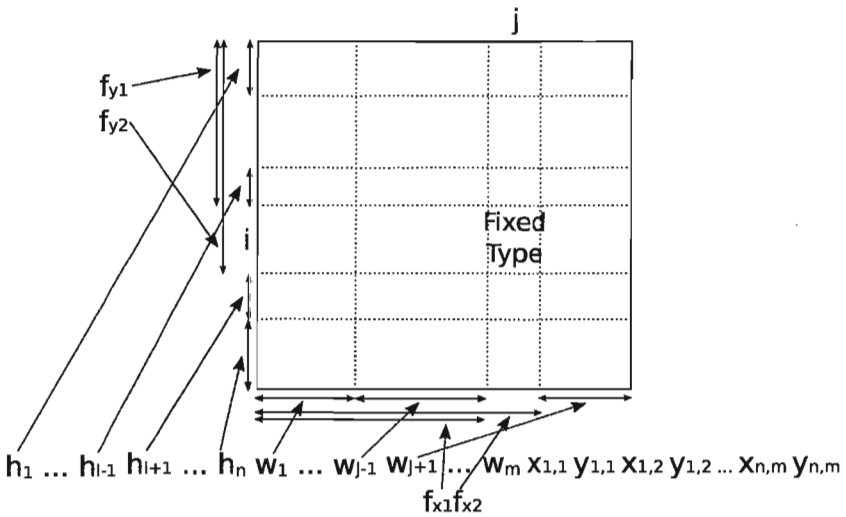
Windows measures how well the window requirements of various room types are satisfied. The objective score is simply the number of rooms that should have windows but cannot because they are internal within the building structure.

4.6 Fixed Rooms

To allow creation of a building with several floors without modifying the chromosome or increasing the complexity in search space, the following strategy is used. A particular room within the floor plan is fixed in position, size and room type. By having a fixed room in the evolved plans any plan for the bottom floor can be combined with any plan for an upper floor to create the building.

Specifically, fixing a room in position is supported in the GA representation as shown in Figure 4.6. The four width and height constraints are guaranteed by adjusting the widths and heights proportionally. There is no obvious way to fix a room within the GP representation, so this was not implemented.

Not only does evolving separate floors separately reduce the size of the search space, but it also allows for a separate requirement specification for each floor. It's not uncommon to have very different needs on certain floors. It's also not uncommon for one floor plan to be repeated several times in apartment buildings or office buildings.



$$\sum_{x=1}^{i-1} w_x = f_{x1}$$

$$\sum_{x=i+1}^{m} w_x = width - f_{x2}$$

$$\sum_{y=1}^{j-1} h_y = f_{y1}$$

$$\sum_{y=j+1}^{n} h_y = height - f_{y2}$$

Figure 4.6: Fixed room representation

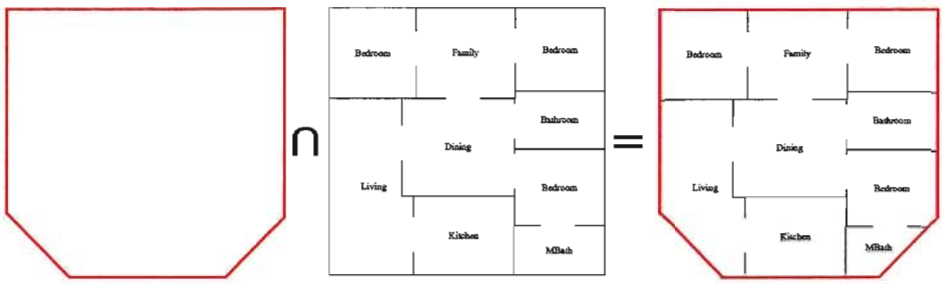


Figure 4.7: Cutting a polygonal shape out

4.7 Polygonal Layouts

Often it is necessary to design a house to fit a polygonal boundary. This may be due to spatial limitations or for aesthetic reasons. The strategy used to construct non-rectangular houses within this thesis is to evolve a house to fit the bounding box of the outer shape required and then clip it to the required shape.

Figure 4.7 shows an example of how a polygonal floor plan is created out of a rectangular one. This is done as a part of the construction process such that removed adjacencies and rooms can be properly accounted for in the fitness of the result. This means that if the clipping removes an important room it will be penalized for that in the resulting fitness.

4.8 Diversity Preservation

A common problem with converging population based evolutionary algorithms is that the entire population may converge onto a single or a few good results. In order to avoid this, a diversity preservation strategy is used in this system. After ranking individuals (or calculated a weighted sum), the population of individuals is scanned for duplicates and the final score or rank of each duplicate individual is given a penalty of $diversity \cdot i$ where $diversity$ is the diversity preservation factor and i is how many times this individual has already been seen earlier in the population.

Chapter 5

Evolution of a Basic Floor Plan

This chapter explores the experimentation of evolutionary parameters and problem specification with respect to their effects on the resulting solutions. Section 5.1 outlines several experiments measuring the relative success of various evolutionary algorithms, parameters, and multi-objective ranking schemes. Section 5.2 considers what is necessary in the specification of requirements for a house in order for its suitability as a house to be successfully captured by the objective functions and satisfactory houses are evolved.

5.1 Evolutionary Parameters

There are a large variety of evolutionary parameters and methods with which to evolve floor plans. A few fundamental experiments are carried out first in order to tune some of the various parameters to be used in the following searches. These experiments will test the effectiveness of various parameters in a basic floor plan.

The floor plan to test various parameters is a simple bungalow style home. It requires one to two bathrooms, at least two bedrooms, one with a master bathroom, a kitchen, a dining room, and a social room. The entire house is built in a 40 by 30 foot rectangle. The fitness is measured by the satisfaction of having the above rules as well as the rooms it creates satisfying certain constraints themselves. The default evolutionary parameters are shown in Table 5.1. 30 runs are performed such that a one-tailed Z-test may be used to show the confidence in one experiment outperforming another. The house parameters are shown as they exist in the program in Listing 5.1.

Table 5.1: Evolutionary Algorithm Parameters

Parameter	Value
# of runs	30
Method	GA
Dimensions	40' x 30'
Population	500
Generations	200
Crossover	80%
Mutation	20%
Reset if stalled	10 generations
Selection Method	Tournament
Tournament Size	3
Assignment	Procedural
Diversity Factor	100
Ranking Method	Diverse Normalized Ranked Sum

Listing 5.1: House requirements for evolutionary experiments

```
Outside
{
  type: public;
  attach: Social Room, Hallway;
}

Social Room
{
  type: public;
  minimum: 1;
  maximum: 1;
  attach: Bathroom, Kitchen, Dining Room, Bedroom, Hallway,
     Social Room;
  min-width: 7;
  min-area: 150;
  max-ratio: 1.5;
  bigger-than: Bedroom, Kitchen, Dining Room;
}

Bathroom {
  type: private;
  minimum: 1;
  maximum: 2;
  min-width: 5;
  min-area: 35;
  max-ratio: 1.5;
  bigger-than: Closet;

  width: 6;
  area: 54;
}

Master Bathroom {
  type: private;
  minimum: 1;
  maximum: 1;
  min-width: 5;
  min-area: 35;
  max-ratio: 1.5;
  bigger-than: Bathroom, Closet;

  width: 7;
  area: 60;
}
```

```
Kitchen {
  type: public;
  minimum: 1;
  maximum: 1;
  windows: yes;
  attach: Dining Room(0-1), Social Room;
  min-width: 8;
  min-area: 100;
  max-ratio: 1.5;
  bigger-than: Bedroom, Bathroom;

  access: Dining Room;
  area: 120;
}

Dining Room {
  type: public;
  minimum: 1;
  maximum: 1;
  attach: Social Room, Kitchen;
  min-width: 9;
  min-area: 100;
  max-ratio: 1.5;
  bigger-than: Bedroom;

  access-nearest: Bathroom;
  area: 120;
}

Bedroom {
  type: private;
  minimum: 2;
  windows: yes;
  attach: Master Bathroom(0-1), Closet(0-1);
  min-width: 7;
  min-area: 80;
  max-ratio: 1.6;
  bigger-than: Bathroom, Closet;

  access-nearest: Bathroom;
  area: 150;
}

Hallway {
```

```
  type: public;
  minimum: 0;
  maximum: 1;
  min-ratio: 2;
  attach: Bedroom, Bathroom, Social Room, Dining Room, Kitchen;
  twists: 3;
}

Closet{
  type: private;
  minimum: 0;
  max-ratio: 3;
}
```

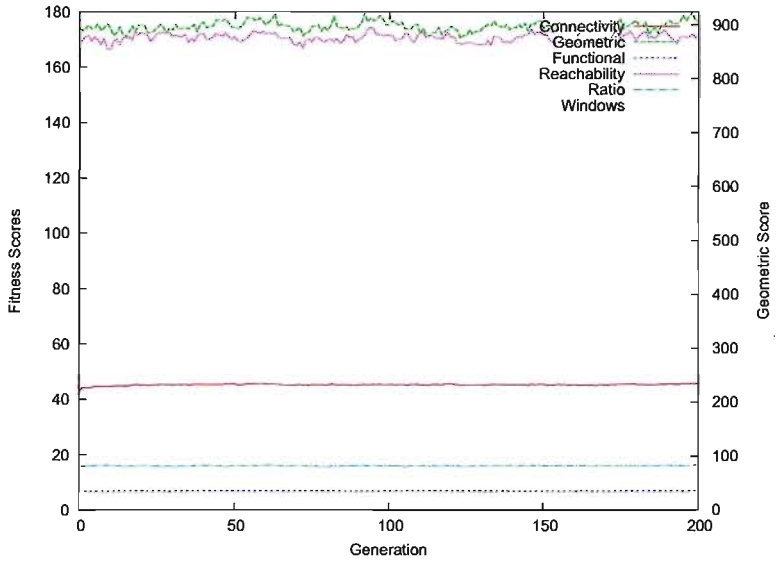
5.1.1 Random Search Vs Evolutionary Pressure

The first test, and probably the most important, is to ensure that the evolutionary pressure is making a difference in the evolution. A simple test is to reduce the tournament size to 1. When doing tournament selection, k individuals are selected and the most fit of those k individuals is the winner. If $k = 1$, then a random individual is selected as the winner. This means that the fitness of the individual is irrelevant in its selection for breeding and is equivalent to performing a random search.

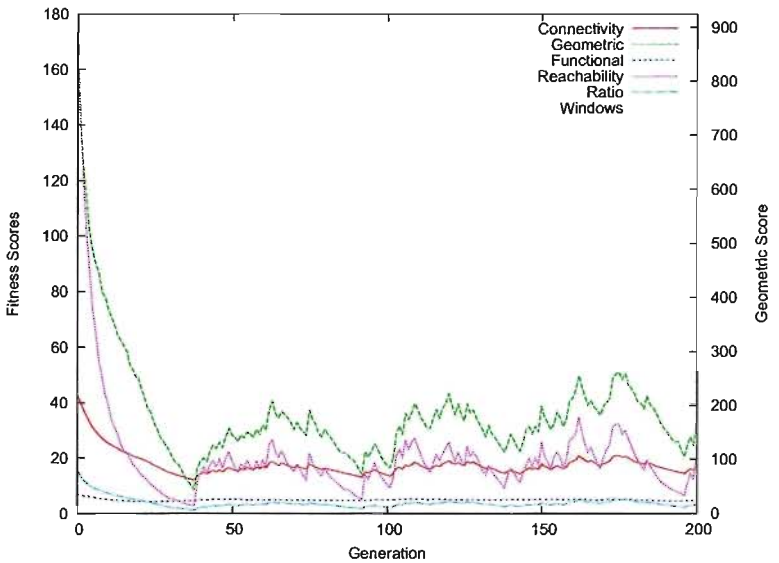
Figure 5.1 shows the results of this test. There is a clear cut difference in the population average although this is to be expected with a random search. Given that it is randomly creating individuals, the average individual will have a poor fitness. The directed search shows that evolutionary pressure and the GA crossover are working to combine good individuals and obtain other good individuals.

Figure 5.2 shows the average of the best individuals in each generation for each of the runs. The first thing to note is the difference in the scales of the axes. The random search has much higher (worse) fitness values in the best individual than the directed search. Additionally it can be observed that while there is some improvement in the best individual at the start of the directed search, the random search doesn't show any sign of improvement. Furthermore, the best scores in the random search are much worse than the directed search.

Table 5.2 shows the average best fitness over 30 runs of the experiment using a random and a directed search. The confidence that the directed

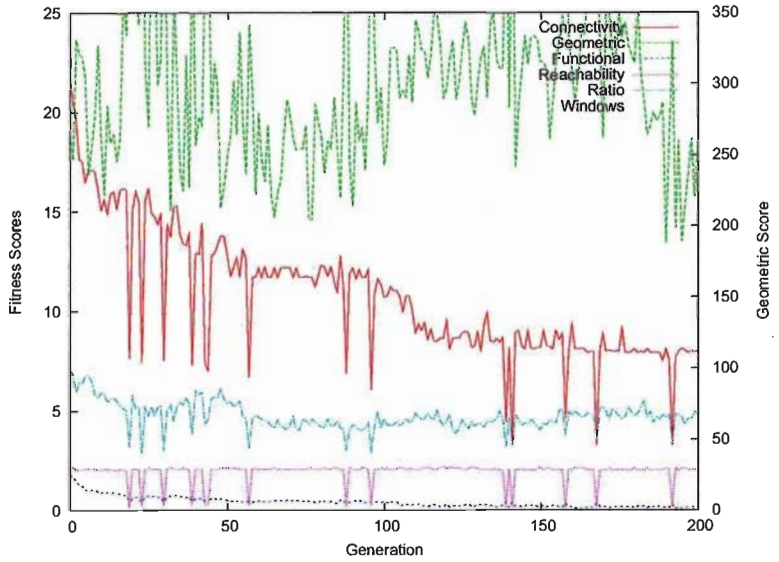


(a) Random Search

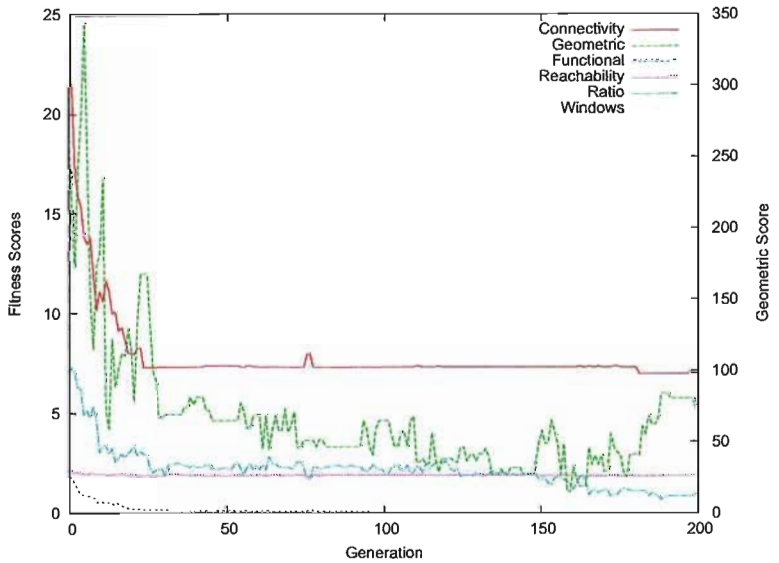


(b) Directed Search

Figure 5.1: Random versus directed search – Population average



(a) Random Search



(b) Directed Search

Figure 5.2: Random versus directed search – Average best individual

Table 5.2: Confidence in Directed Search Vs. Random Search

Average Best Fitness						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
Random	**8.0**	**260.3**	**0.1667**	**2.034**	**4.768**	0.03333
Directed	7.0	72.06	0.0	1.904	1.063	0.0
Confidence	95.4%	99.2%	99.3%	99.9%	99.9%	–

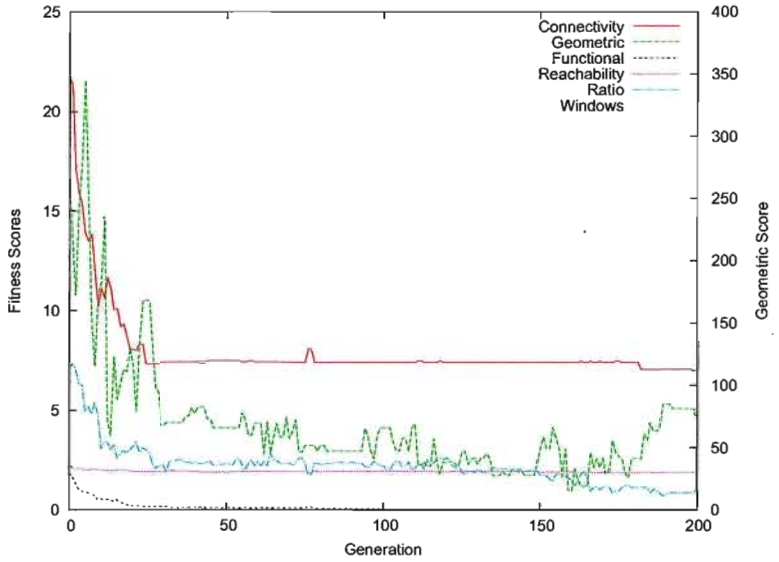
search is better (lower) than the random search is calculated using a one-tailed Z-test. As can be seen, the directed search achieves a better score across all of the objectives with a confidence of at least 84%. The window objective score is a fairly easy objective to satisfy which is why the random search was able to do relatively well.

5.1.2 Genetic Algorithm Vs Genetic Programming

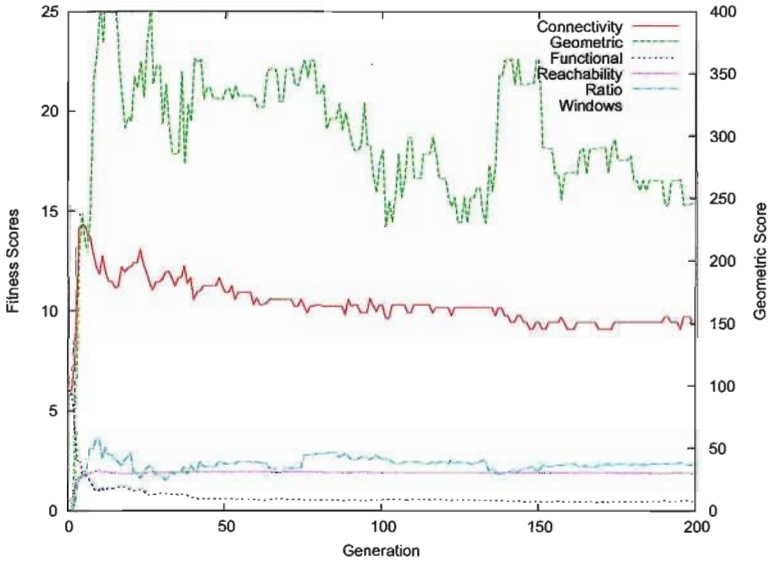
One of the major focuses of this research is to compare a fixed size genetic algorithm representation of a house with a dynamically sized tree-based divisive representation of a house. The results can be compared via fitness graphs to see how they work in an evolutionary context.

Figure 5.3 shows the comparison of the genetic algorithm to the genetic program method. The GA shows overall convergence in the population towards better fitness values whereas the GP fails to advance much beyond its initial exploration. The most likely explanation is that the crossover operator in the genetic program is destructive with respect to the "good" aspects of the two parent individuals it is trying to preserve. When the GP inserts a branch from one individual into another it may generate a very different floor layout depending on where in that other individual the branch is inserted.

Table 5.3 shows a statistical analysis comparing the results from the genetic programming approach to the genetic algorithm. As can be seen, the genetic algorithm outperforms the GP in nearly all of the fitness objective values with a high degree of confidence. Both algorithms are able to completely optimize the Window objective further showing that it is an easy one to satisfy.



(a) Genetic Algorithm



(b) Genetic Program

Figure 5.3: Comparing evolutionary methods – Average best solution

Table 5.3: Confidence in Genetic Algorithm Vs. Genetic Programming approach

Average best fitness						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
GA	**7.0**	**72.06**	**0.0**	1.904	**1.063**	0.0
GP	9.333	245.2	0.4667	**1.866**	2.299	0.0
Confidence	99.9%	99.4%	99.9%	94.4%	96.0%	–

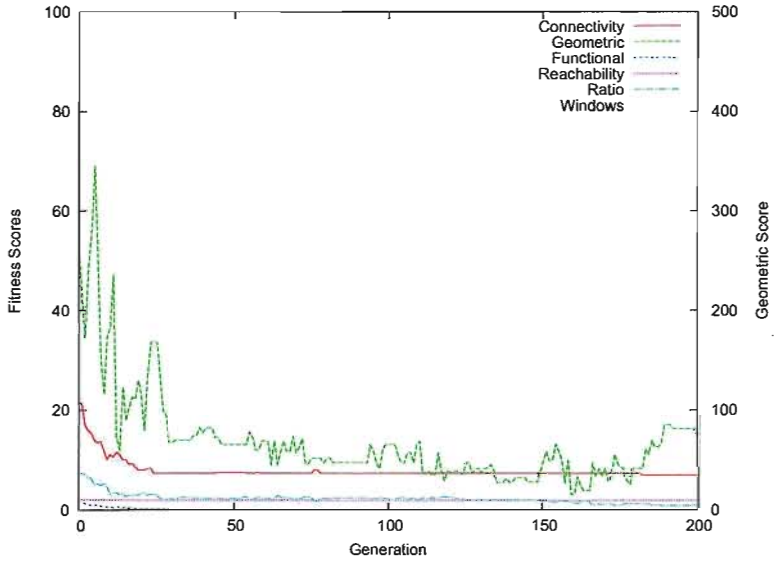
Table 5.4: Confidence in Procedural Assignment Vs. Evolutionary

Average best fitness						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
Procedural	7.0	72.06	0.0	**1.904**	1.063	0.0
Evolutionary	7.417	**3.584**	0.0	54.43	0.6798	0.0
Confidence	–	99.2%	–	99.9%	–	–

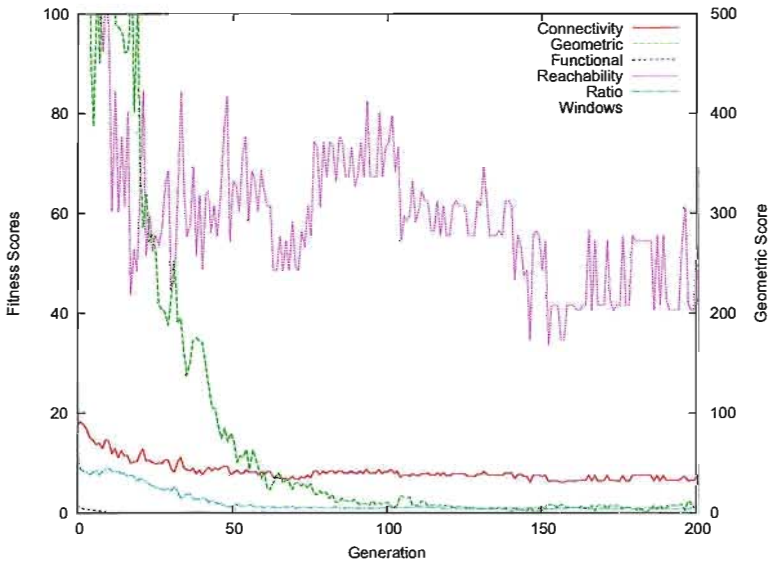
5.1.3 Procedural Vs Evolutionary Assignment

Two assignment strategies have been outlined in this thesis. A procedural one which basically delegates the responsibility of room assignment to be part of the transformation of genotype to phenotype and an evolutionary one in which the room types are part of the genotype. The plans produced by the procedural assignment have less flexibility and a much smaller search space which can drastically improve the scalability of such a system.

Figure 5.4 shows a comparison of the average best fitness using a procedural and an evolutionary assignment strategy. Table 5.4 shows the average best fitnesses. The evolutionary algorithm outperforms the procedural algorithm in the Geometric and Ratio scores, however the procedural algorithm performs far better in terms of Reachability which is much more important for the overall functionality of the house. Figure 5.5 shows the kind of disconnected houses that the evolutionary assignment algorithm produces and how this is reflected in the reachable objective.

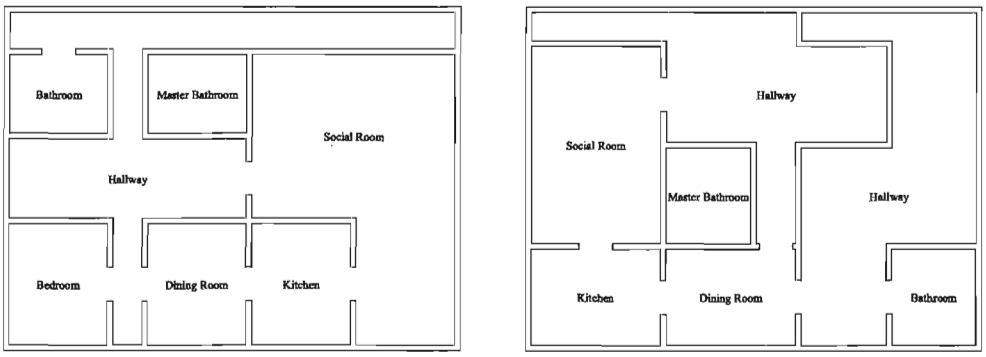


(a) Procedural assignment



(b) Evolutionary assignment

Figure 5.4: Comparing assignment strategy – Average best fitness



Reachable: 210 Reachable: 210

Figure 5.5: Examples of disconnected rooms produced by evolutionary assignment

5.1.4 Comparison of Multi-Objective Ranking Strategies

There are many conflicting and even independent objectives to be optimized in the creation of a floor plan. As discussed earlier there are a variety of means by which to attempt to optimize them. Pareto ranking has long been the standard approach to multiobjective optimization problems, but it does not scale well with many objectives. In order to deal with a greater number of objectives ranked sum has been implemented, but with the widespread use of pareto ranking it is prudent to compare the two and see if it improves the results. Ranked sum is used to select a single best individual from the population of rank 0's when using Pareto ranking.

A common problem with either ranking method is convergence and so each shall be tested with and without a diversity preservation strategy. The diversity preservation is that in the event that two individuals are identical, one will receive a penalty to its overall rank as described in Section 4.8. In this manner the algorithm can be prevented from converging onto a few results.

Table 5.5 shows a statical analysis of all of the multi-objective strategies. D and N stand for diverse and normalized respectively. The diverse experiments have a diversity factor of 100 whereas the others have no penalty for identical individuals in the population. In general, using both diversity

Table 5.5: Statistical comparison of multi-objective ranking strategies
D = Diversity preservation, N = Normalized

Average best fitness						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
Weighted	7.333	0.0	0.1667	1.827	0.0020	0.0
Pareto	7.333	13.97	0.0333	1.9	1.462	0.0
Ranked	7.0	0.0	0.0	1.875	0.2073	0.0
N. Rank.	7.0	0.0	0.0	1.879	0.0	0.0
D. Weighted	7.533	0.0	0.1333	1.789	0.2233	0.0
D. Pareto	7.0	72.06	0.0	1.904	1.063	0.0
D. Ranked	7.0	0.0	0.0	1.882	0.2016	0.0
D. N. Ranked	7.0	0.0	0.0	1.877	0.0	0.0

Confidence in Diverse Normalized Ranked over other methods:						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
Weighted	99.3%	–	99.3%	–	84.6%	–
Pareto	84.6%	90.9%	84.6%	94.2%	99.9%	–
Ranked	–	–	–	–	92.8%	–
N. Rank.	–	–	–	–	–	–
D. Weighted	99.9%	–	98.4%	–	94.9%	–
D. Pareto	–	99.5%	–	96.7%	99.9%	–
D. Ranked	–	–	–	88.9%	95.5%	–

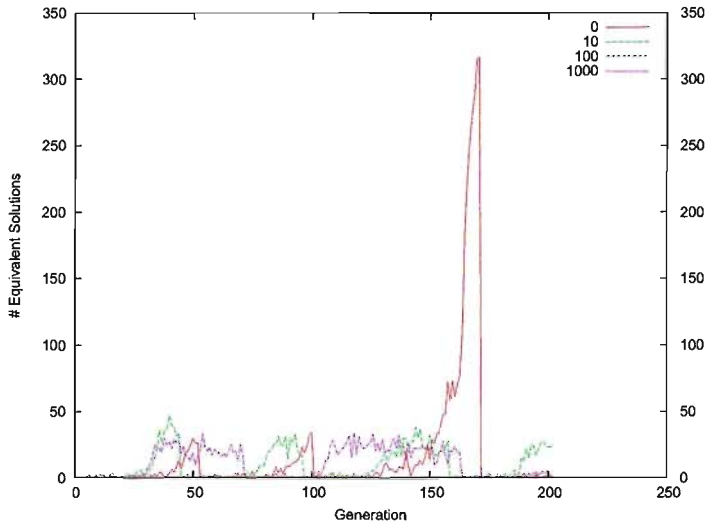


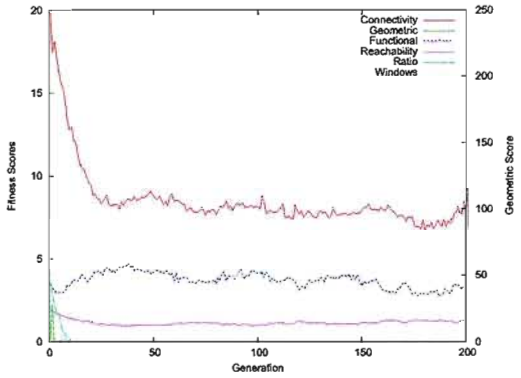
Figure 5.6: Number of identical individuals with and without diversity preservation.

preservation and normalized ranking produces objective values that are as good if not better than all other tested strategies. There are a few exceptions to this, however in these cases other fitness values were sacrificed.

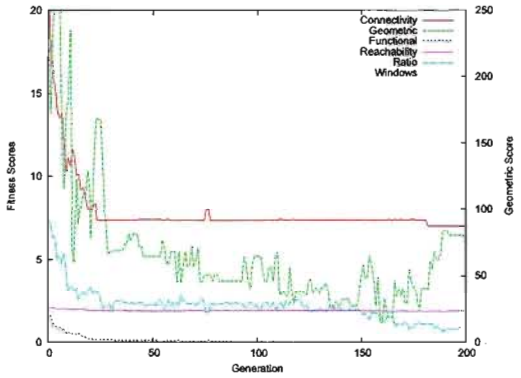
5.1.5 Diversity Preservation

The use of diversity preservation strategies was tested in conjunction with multi-objective ranking strategies in the last step, however in this section the diversity preservation factor is tested. In order to get an idea of how diverse the population is, the numbers of unique individuals, ranks and the distribution of these ranks are measured throughout several experiments over 30 runs.

Figure 5.6 shows the number of identical houses in the population on a random run with various levels of diversity preservation. Without diversity preservation, when the system converges it tends to fill the population with many copies of the same house. Once diversity preservation is used it does not create nearly as many identical individuals as they cannot persist through the generations. This also means far fewer identical houses in the resulting population so that the user has a variety to choose from.



(a) Equal weighting



(b) Functional weighted most important

Figure 5.7: Effect of weighting on average of best fitnesses

5.1.6 Objective Weighting

Using a multiobjectve ranking scheme like ranked sum has the effect of optimizing all objectives equally. In this problem, as in others, some objectives are more important to optimize than others. For example a house that is missing key rooms like a bathroom or a kitchen will not be very usable. When combining the ranks using ranked sum, weighting can be imposed by multiplying the ranks by a weight value.

Figure 5.7 shows the effect of weighting the functional objective as being more important than the others. Table 5.6 shows the average best fitnesses and the confidence in the weighted runs outperforming the equally weighted

Table 5.6: Confidence in weighting objective values Vs. equal weighting

Average best fitness						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
Equal	7.833	**0.0**	3.233	**1.274**	**0.0**	0.0
Weighted	**7.0**	72.06	**0.0**	1.904	1.063	0.0
Confidence	98.6%	99.5%	99.9%	99.9%	99.9%	–

runs. The effect of giving the Functional objective a weight of 10 is that the functional is almost guaranteed to be improved over having an equal weighting. The connectivity score seems to improve indirectly as a result while the other objectives are noticeably compromised. However, given that the Functional score is one of the most important ones this is a desirable compromise.

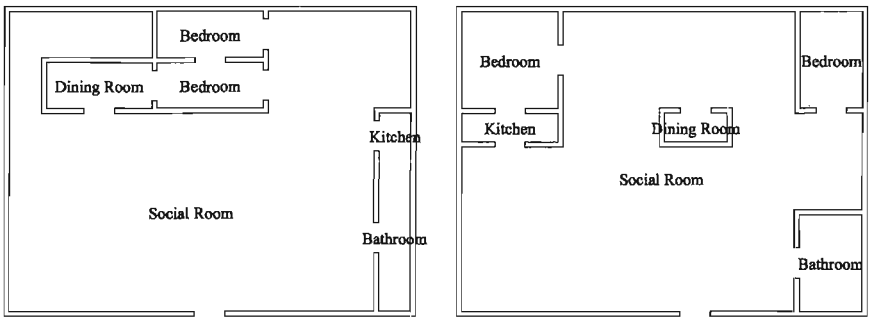
5.2 Requirement Specifications

When talking about the design of houses, a lot of details are often taken for granted. In this section this concept will be analyzed in detail. A common starting point when stating the design of a house (given that the dimensions have already been fixed) are which rooms and how many will be in the house. This shall be the starting point.

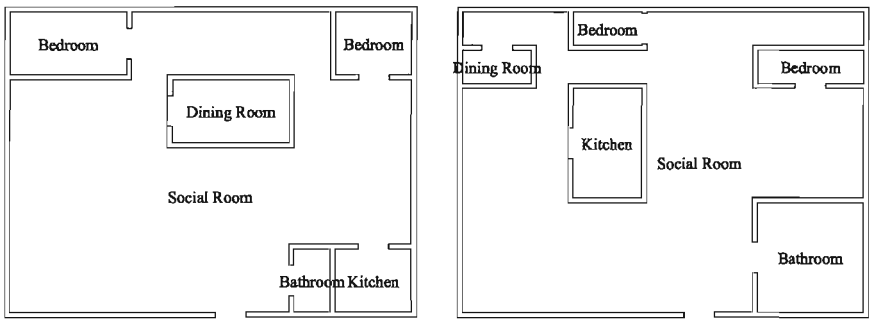
5.2.1 Basic Functionality and Connectivity

Assume the customer in question wants a house that contains one social room, one bathroom, a kitchen, a dining room, a bedroom and no more than one hallway. Given that the purpose of this section is to start as generic as possible there will be no impositions of which rooms can attach to which rooms, and any room shall be capable of attaching to any other room. This would restrict the system as little as possible in its satisfaction of the requirements.

Looking at Figure 5.8(a), it's quite evident what is wrong with this requirement specification. The system has no contextual information about



(a) Number of rooms only



(b) Connection restrictions between rooms

Figure 5.8: A selection of houses evolved with basic requirements.

what a bedroom is, or how it is different from a social room or bathroom. While it could have in theory made a dream house, there would be no way of objectively recognizing that the dream house is any better than the others. Additionally one must consider the odds of producing such a house when anything with rectangular walls can be created. Restrictions and direction are necessary in order for the system to produce sensible layouts.

One noteworthy restriction is that one would expect a certain flow of rooms. Rather than being able to move from any room to any room there is a certain expectation of which rooms will be entered from other rooms. This relates to the significance of the justified gamma map described in Section 3.1 in defining the flow of a house.

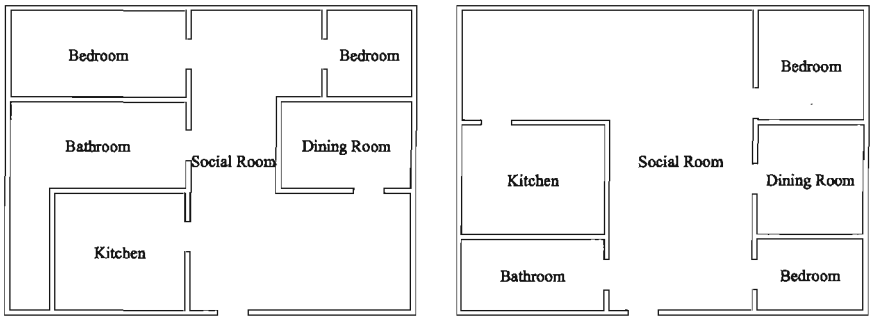
Common sense would dictate that one enters a house into a social room or a hallway. A social room or hallway lead to any room type, whereas a kitchen may lead to a dining room or a social room. Similarly a dining room might lead to a kitchen or another social room. Given that this problem is not concerned with generating a master bathroom, a bedroom or bathroom would not lead anywhere further. When these requirements are specified the differences become apparent in the resulting houses. Figure 5.8(b) shows the results. The flow of rooms from the entrance through the house makes more sense.

5.2.2 Room sizes and ratios

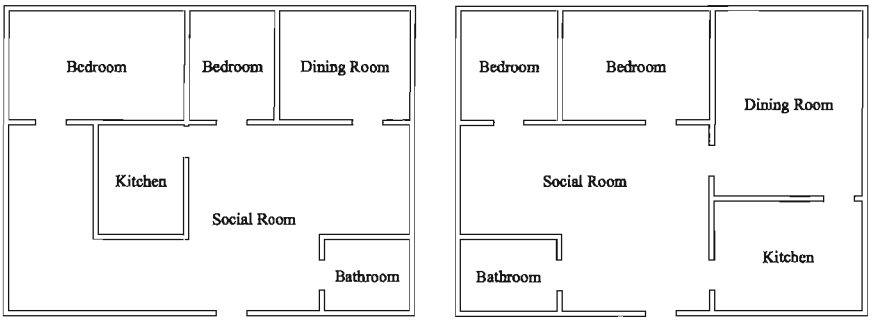
In the previous section a house was evolved with the necessary rooms and only sensible connections between rooms. However, it is still evident that the system needs some contextual information as to the required sizes of rooms. While these are typically dictated by the items commonly stored in the room, since the system is not concerned with the placement of furniture it needs to know the size requirements of the rooms.

Some minimum room areas will be given such that the resulting house will have enough room to be appropriately furnished. A social room must be at least 140 square feet, a kitchen at least 90, a dining room at least 100, and a bedroom at least 80. Additionally minimum widths are given for each of the rooms as there are certain length objects that must fit. The resulting floor plans in Figure 5.9(a) are beginning to look much more sensible. Though there are still some oddities like long narrow sections of rooms.

In order to rectify this a maximum ratio is specified. This is measured as the larger of width over length or length over width of any rectangular



(a) Minimum room area



(b) Maximum ratios

Figure 5.9: Area and ratio requirements in basic layout.

region in the room. A maximum ratio of 1.5 is given. The resulting houses in Figure 5.9(b) have more regular ratios. The layouts seem more natural, though there are still odd cases where a bathroom is bigger than a bedroom. By the fitness measurement this is perfectly acceptable as they are both larger than their minimum areas but visually and functionally it seems wrong.

5.2.3 Size Relations

In order to make the floor plans seem more sensible it would make sense to impose size relations on the rooms. In this case, social rooms should be bigger than kitchens and dining rooms which are bigger than bedrooms which are bigger than bathrooms. By enforcing these size relations the floor plans should look more sensible.

Figure 5.10 shows the resulting floor plans with imposed size relations. With this constraint the system filters out all floor plans which violate the imposed relations and the remaining houses seem more logical in their design.

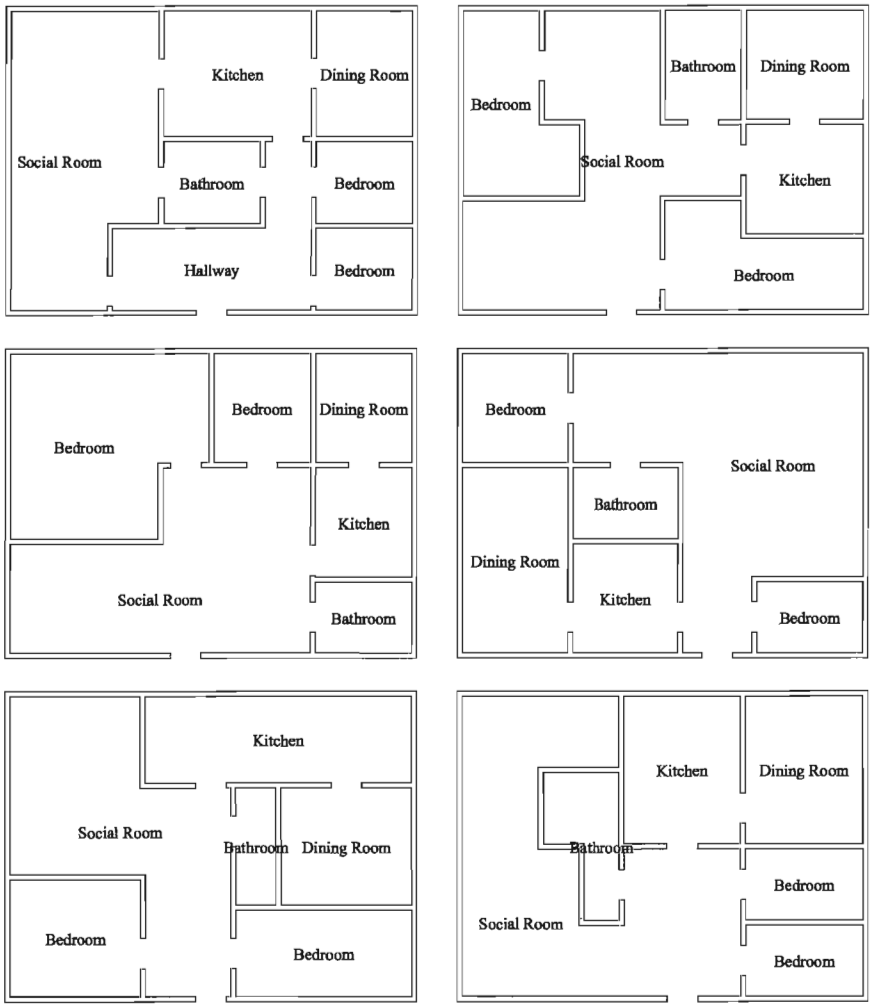


Figure 5.10: Basic two bedroom bungalow with size relation constraint enforced

Chapter 6

Advanced Floor Plans

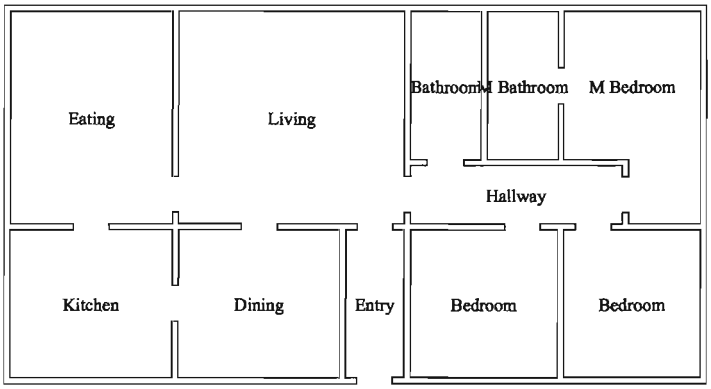
This chapter explores some of the more advanced capabilities of this system. A house from a commercial plan book is reconstructed and then used as a target for evolution in Section 6.1 to compare how well the system can do with similar requirements. Polygonal shaped houses are constructed in Section 6.2. In Section 6.3, office buildings are evolved as an example of an alternative application. Section 6.4 extends this to support multiple floor office buildings. Lastly, Section 6.5 explores using the system for the construction of grocery store layouts.

All of these experiments use the established evolutionary settings determined in Section 5.1 and listed in Table 5.1 as these parameters have already been shown to work well in smaller experiments.

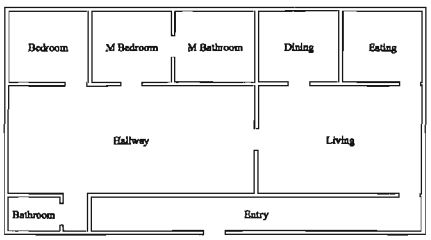
6.1 Comparison to a real floor plan

To compare the quality of solutions generated to those from a real architectural book, a floor plan has been selected from a commercial book with stock contemporary house plans[26] and loosely reconstructed in the representation used by the GA. This reproduction is shown in Figure 6.1(a).

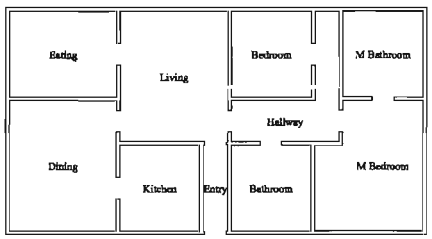
The key features taken from this floor plan and specified to the system are the numbers of rooms, and the overall idea that the main rooms are attached to the living room while the bedrooms and bathroom are accessible from a hallway. A full list of the specifications can be found in Listing B.1 in the appendix. The reconstructed floor plan's fitness with respect to the specified requirements is detailed in Listing D.1. The master bedroom is too narrow



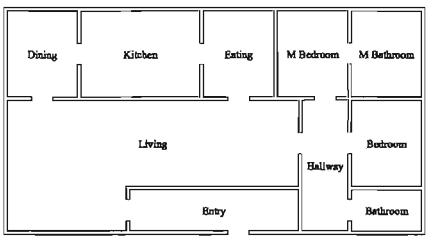
(a) Recreation of real floor plan in GA format



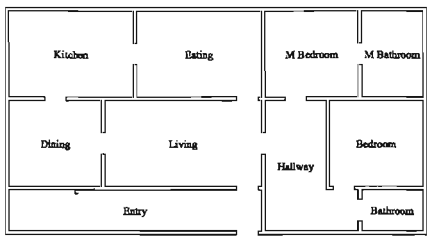
(b)



(c)



(d)



(e)

Figure 6.1: Comparison of a real floor plan to evolved houses with similar requirements.

Table 6.1: Fitness of original and evolved "real" floor plans in Figure 6.1

House	Original	House b	House c	House d	House e
Connectivity	9	9	9	9	9
Geometric	12.642	0	0	0	0
Functional	0	1	0	0	0
Reachable	3.08333	2.9	3	3	3
Ratio	5.98098	0	0	2.65085	2.79901
Windows	0	0	0	0	0

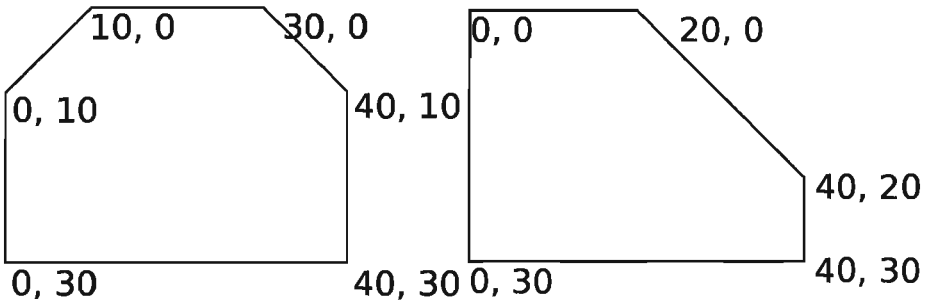
in the corner which leads to the bad geometric score even though the room is mostly large enough. Additionally the bathrooms are fairly long and narrow which is punished in the ratio score.

Figure 6.1 shows a selection of the evolved houses using the same number of grid cells and dimensions as the reconstructed plan. Given that a very specific ordering of rooms was imposed upon the system the variety of plans generated is somewhat limited. However, most of the generated houses have the same spirit as the reconstructed plan. The complete fitness score of all the plans shown is listed in Table 6.1. House (b) did not manage to place a kitchen which led it to have a non-zero functional score. Otherwise, the evolved plans typically exceed the scores of the hand constructed house. This does not mean the hand constructed house is bad, it just shows how the fitness score does not necessarily mean everything in terms of the house quality. Generally a reasonably good fitness means there is nothing wrong with the house.

6.2 Polygonal houses

In Section 4.7 a strategy to evolve polygonal houses is described. The house is evolved to fit the bounding box of the house shape, and clipped to the polygonal region. In this section, this strategy is used to construct a variety of irregularly shaped houses.

A half-hexagon shape is used to construct houses with an interesting exterior appeal as shown in Figure 6.2(a). A 40' by 30' house is evolved and then clipped to this shape. Figure 6.3 shows several examples of the houses



(a) Half hexagon shape (b) Triangular shape

Figure 6.2: Shapes used for polygonal experiments

the resulted from this search. There are some noticeable deficiencies in some of the designs. For example, house (b) has a useless hallway from the social room leading to the bedroom. In the current fitness specification this is not directly punished. Indirectly it is discouraged as there is less room for the remaining rooms in the house. House (d) on the other hand has a rather large hallway with a corridor that leads nowhere on the bottom-left of the plan. Again, this is not directly punished as a hallway is allowed and it does connect other rooms. It is only punished indirectly in that the kitchen could have been larger. Overall the house plans seem functional and quite acceptable.

To demonstrate the feasibility of these houses, Figure 6.5 shows what the plan from house (b) could look like furnished. It has been constructed in Google Sketchup[3] and furnished using public domain models[2]. The extra hallway was removed as could have been done by a post-correction phase or evolved out with more time. With the furnishings, one can see how feasible the house would be. There is a pleasing flow of traffic through the house, the plumbing is largely centralized and all of the rooms are of adequate size for their requirements.

The triangular shape in Figure 6.2(b) is used to construct a variety of houses with a long angular side. Figure 6.4 shows several houses clipped to this shape. In this experiment most of the evolved houses do not make use of their top-right cell in their chromosomes. This is one way in which some garbage data in the chromosome can be introduced, although in the GA representation it is restricted in size and cannot become bloated. One issue that presents itself in several houses is that the width of the room

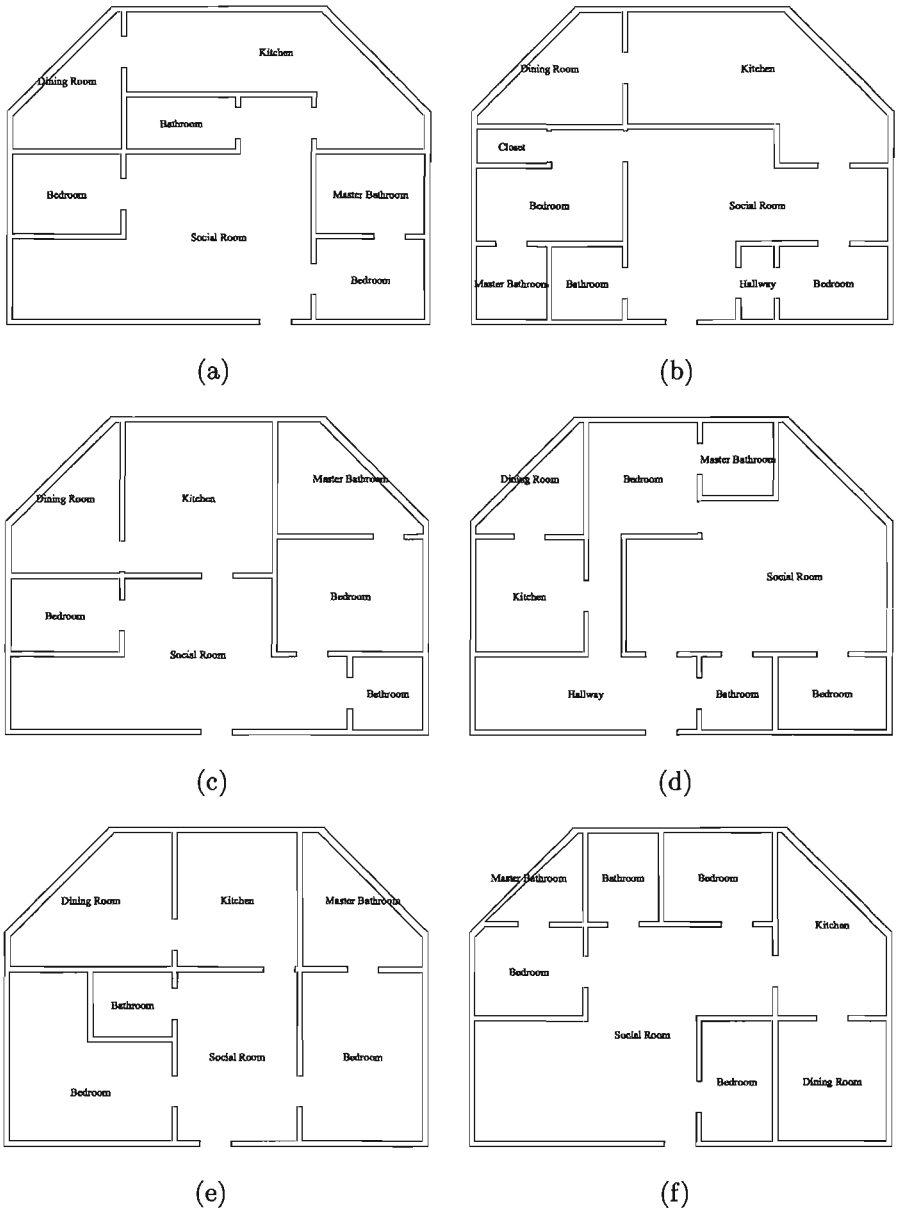
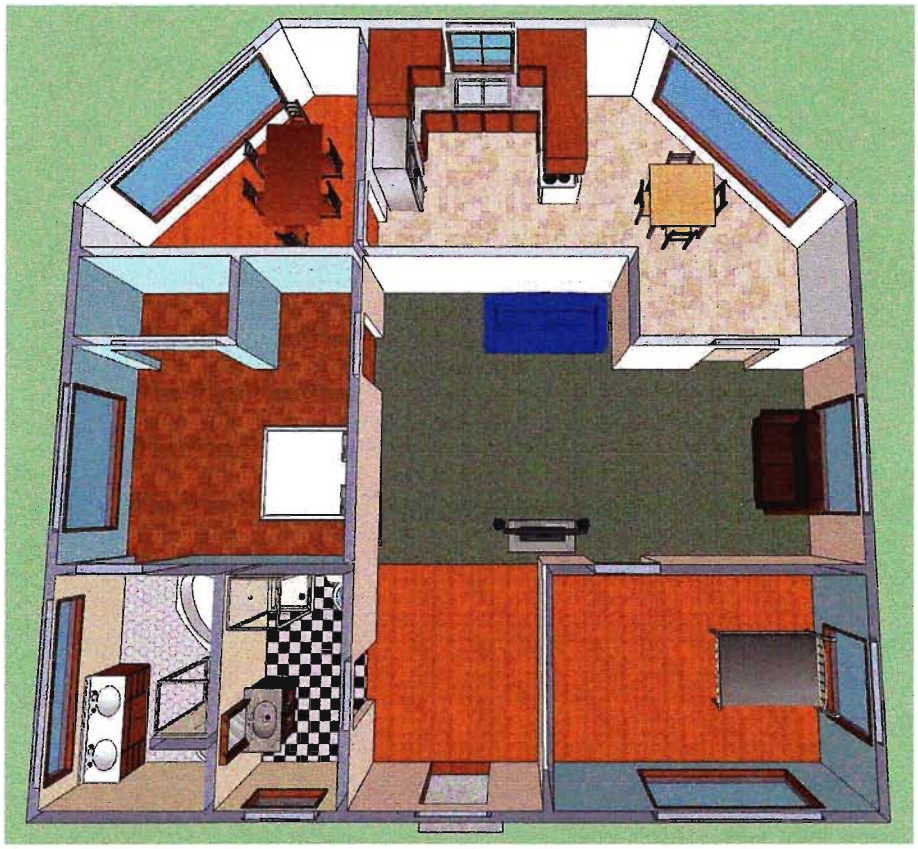
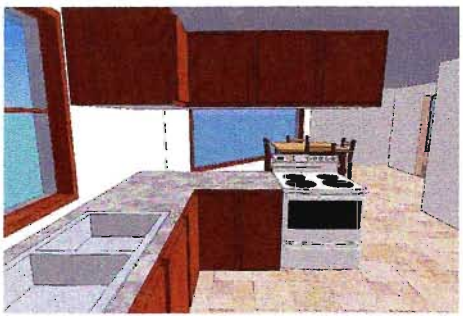


Figure 6.3: Half hexagon shaped houses evolved from system

Figure 6.4: Angle shaped houses evolved from system



(a) Top down view of house with roof removed



(b) View from kitchen

(c) Outside the house

Figure 6.5: 3D view of house from Figure 6.3(b)

is not accurately recognized. This could be solved with various algorithms to analyze shortest widths across arbitrary polygons, but without this the system does not know when rooms are much skinnier or smaller after clipping. House (b) has an example of a kitchen with a very narrow passageway to part of the room. House (c) has a rather small kitchen due to the clipping but it was not measured afterwards. These issues could be easily hand-corrected if desired. Nevertheless, the system produces interesting houses and manages to cope with a long sloped wall for one corner of the house. Many of the designs would make for interesting concepts given this exterior shape.

There are a few weaknesses in the form that the polygonal clipping is currently implemented however. For example, if a room is cut into two rooms as a result of clipping it to the boundary the current clipping algorithm does not currently recognize this. This is not a weakness of the strategy however, just the current implementation of it. The main weakness with the polygonal clipping is that the extra space around the house area is still part of the chromosome and hence somewhat useless. This means the system has a larger search space than necessary as part of the chromosome is essentially bloat. For shapes that fill most of their bounding rectangle however, there is typically no bloat. All of the cells in the matrix end up being used for a room present in the final house layout.

This strategy is a very simple yet powerful method of evolving interestingly shaped houses. While house exteriors often have angled walls, it is usually wasteful to have angled interior walls. By combining a polygonal exterior with rectilinear interior walls a small search space of houses with special shapes can be evolved. The polygonal clipping strategy could even be used to combine the work from this system with another which designs the exterior. House exteriors can be designed separately and then pass the exterior shape on to this system for the interior construction.

6.3 Office building

Since the specifications for the problem give the complete set of rooms, and all of the requirements that describe those rooms, this system can be used for other problem domains with similar requirements. One such example is that of an office building. An office building needs to have offices easily accessible via some system of hallways. Bathrooms need to be within easy reach of those offices and often times there need to be labs and reception areas.

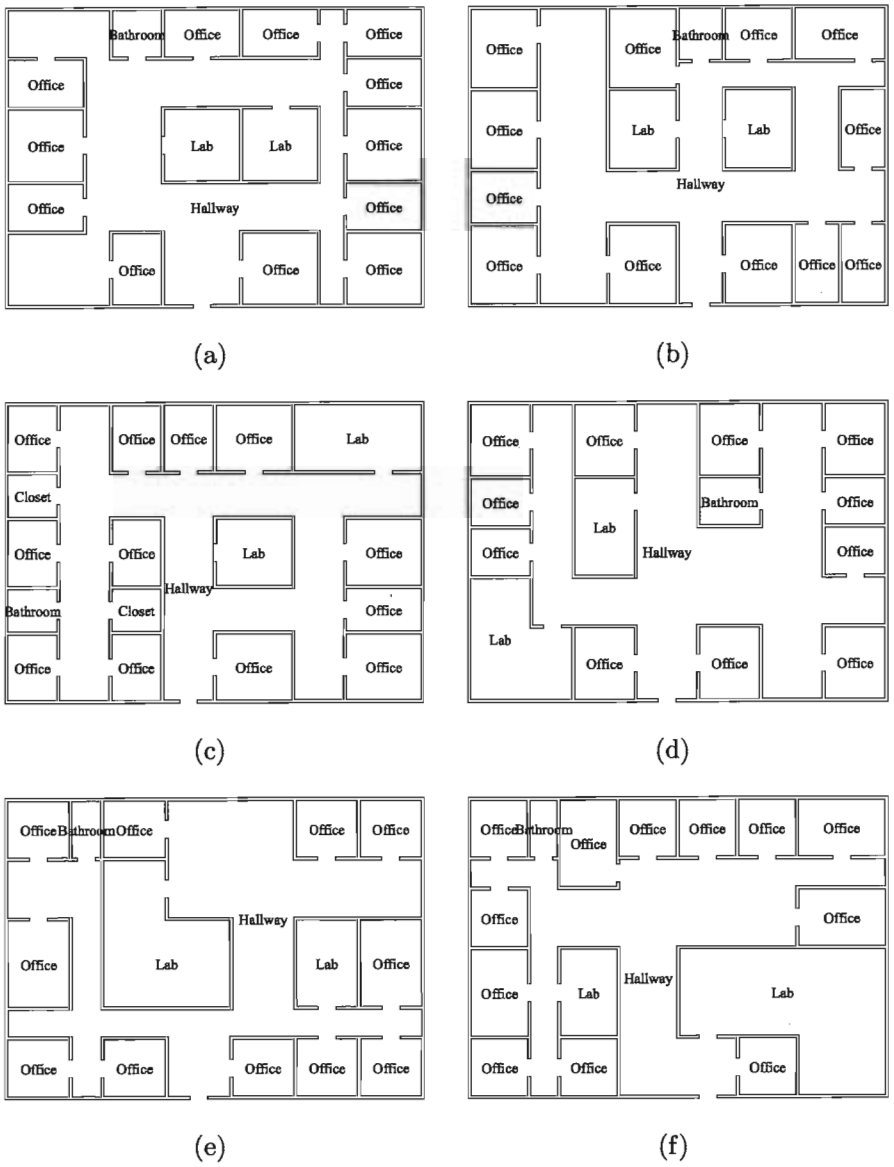


Figure 6.6: A sample of evolved office buildings

The office buildings evolved in this section attempt to construct an easily traversable building with 12 offices, 2 labs, and at least one bathroom. Labs are larger than offices, which are larger than than bathrooms. The goal is to have a single hallway connect all of these rooms to keep the layout simple. An office building requires many more rooms and more intricate designs which necessitates using a larger grid in the chromosome representation. The office building is layed out on a 7 by 5 grid allowing for up to 35 rooms, although in practicality a hallway connecting all of these rooms will require at least 14 or 15 cells on the grid. In terms of layout efficiency the requirements only specify that one must be able to quickly get to a bathroom from the offices.

The complete requirements used for the office buildings are in Listing B.2 in the appendix. A 70' by 50' building is evolved using the same general experiment settings. Figure 6.6 shows a sample of the resulting office buildings. The system manages to evolve intricate single room hallways that span the entire building with no assistance which is very promising for the effectiveness of genetic algorithms in this problem. The office buildings themselves are also very promising, especially given that they have all been generated from a list of basic requirements for the building.

There are a few interesting oddities that the system has evolved in some of the office plans in Figure 6.6. Building (a) has a large room-sized area in the bottom left and top-left of the layout. Similarly, building (e) has a hallway leading to nowhere near the bottom left. On human examination, these irregularities immediately stand out but it is not something that is easy to quantify or measure given arbitrary polygonal shapes. Building (c) on the other hand has evolved a pleasing layout with regular narrow hallways, an adequate number of offices and even a few closet spaces. It is also difficult to quantify what constitutes a good hallway.

In general the evolved layouts have a difficult time obtaining the desired number of rooms. In this larger example, the room requirement was expected to be difficult to obtain and meant to encourage the system to construct as many offices as possible. Most of the final office buildings came acceptably close to fulfilling the necessary room requirements, despite several buildings missing a lab. Nevertheless, many of the final office buildings have nearly all the necessary rooms allocated. Such large and complex specifications are a challenge to fulfill, and so the results obtained are impressive, considering the complexity involved. Furthermore, the offices are quite fit with respect to the other objectives. Offices are fairly rectangular, easy to reach, and washrooms are readily available.

6.4 Multiple floor office building

One of the ways of doing a multiple floor layout is to fix a certain room in position. For example a stairwell or other such room could be given an exact position such that no matter the rooms evolved around it this stairwell would be in the same position on each floor and could thus be combined with any other layout for the other floors.

This strategy is used to evolve a potential multiple floor office building. Any of the plans for a bottom floor from Figure 6.7 can be combined with any of the plans for an upper floor from Figure 6.8. The plans evolved in this experiment required corrections afterwards due to limitations in the specification format. A different post correction step was needed for the bottom floor and upper floor plans.

By having a stairwell room type that could be connected to from a hallway, the system was free to evolve several of these on the bottom floor even though this was punished in the functional score. As a result, the bottom floor created several stairwells before reaching the forced stairwell at the fixed position specified to the program. The post correction applied was to replace all of these extra stairwells with offices as many offices were required in this problem.

On the upper floors, the specification states that a stairwell connects to a hallway. While a hallway cannot connect to other hallways the system was allowed to connect as many hallways as it could to the stairwell where the room assignment started from. The upper floors constructed several separate hallways coming from each side of the stairwell. The post correction applied was to join all adjacent hallways to become one large hallway. Such a post correction could be made automatically or fixed by allowing better generation restrictions such as only one hallway coming from the stairwell.

After corrections the produced plans look pretty decent. The bottom-floor building (a) in Figure 6.7 demonstrates another example of a hallway which does not quite lead anywhere. The upper floors tended to generate rather sparse layouts given that they could construct four separate hallways. Buildings (a) and (c) in Figure 6.8 show examples of this. The space around the stairwell is almost entirely hallway in both of these buildings.

Typically in multiple floor buildings a single plan is repeated over several floors, which means that a selected upper-level plan may be repeated for several floors, followed by another plan. This is one strategy for dealing with evolving plans for multiple floors. This strategy could also be used with

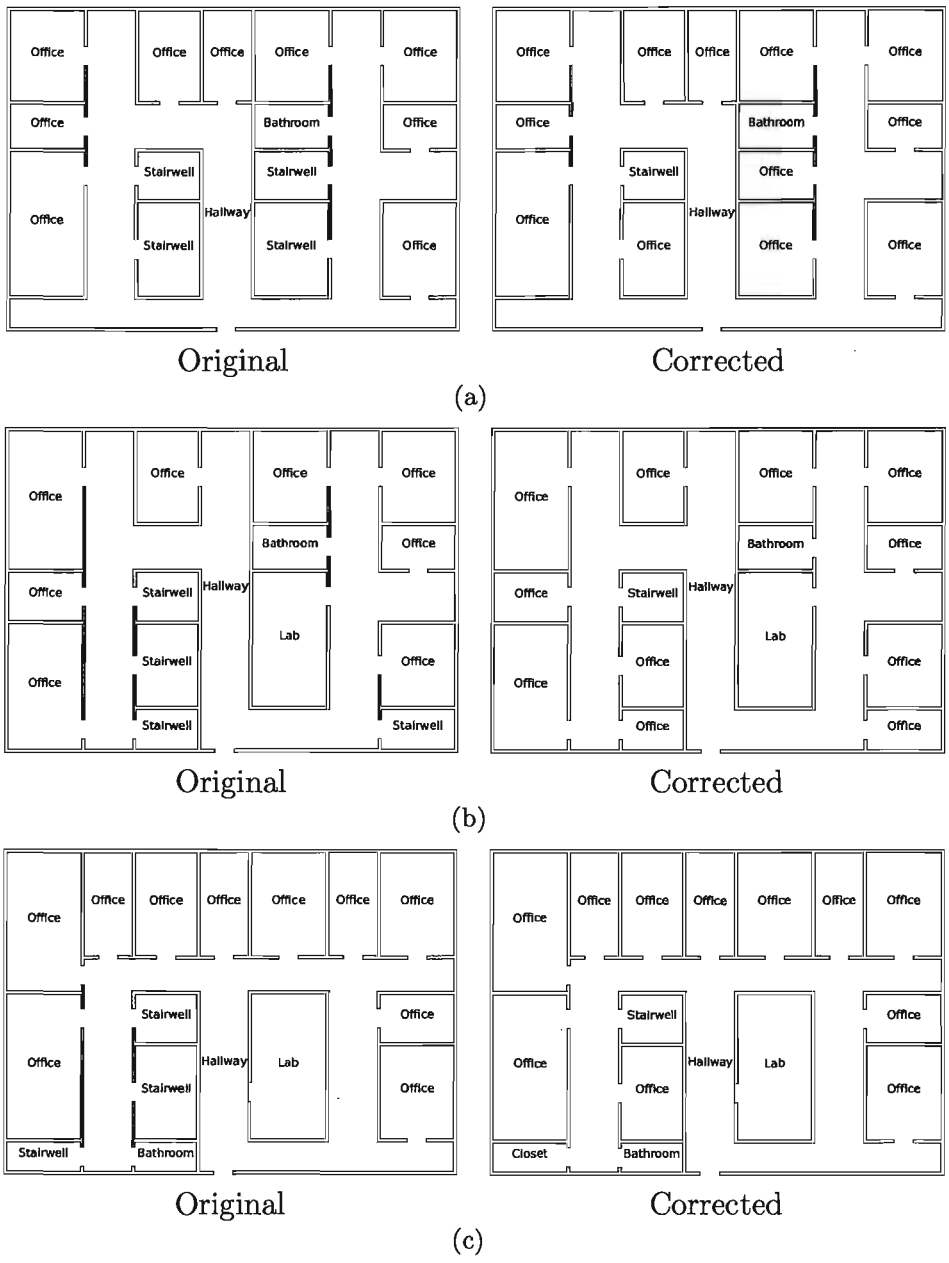


Figure 6.7: Bottom floor office buildings evolved with a fixed stairwell

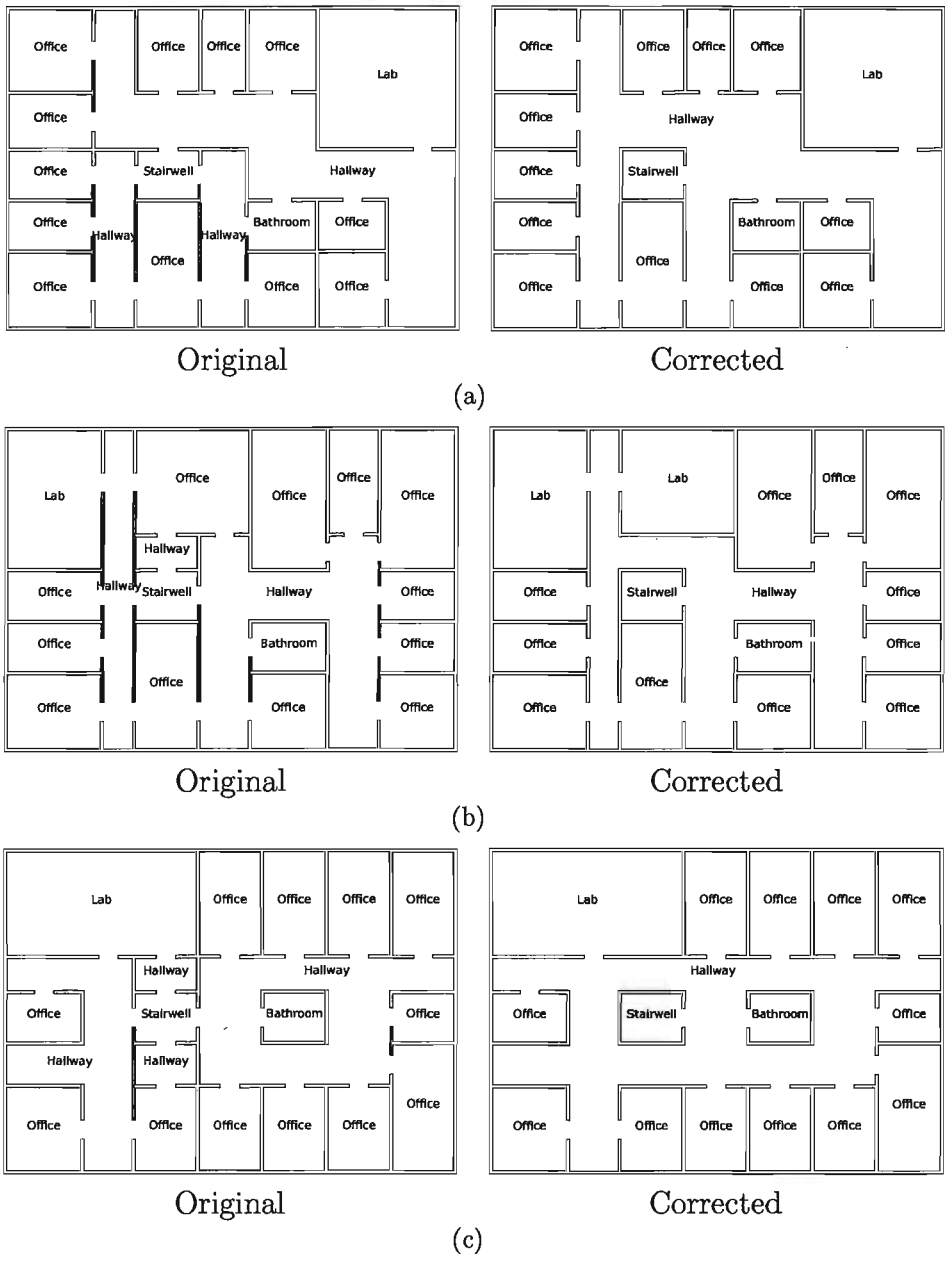


Figure 6.8: Upper floor office buildings evolved with a fixed stairwell

a variety of polygonal shapes to mimic a curved office building, or varying smaller sizes to construct a tall skyscraper that narrows towards the top. The possibilities and combinations are virtually endless. Various floors may have differing requirements, as office buildings often do. Or a composite system could be developed to design the requirements for the various floors of a large scale office building to reduce the amount of work required on the part of a human specifying all of these requirements.

6.5 Grocery Store

Another interesting application is in the layout of something that has discrete regions even though they are not subdivided by walls. This is the case with a grocery store, where there are sections that provide various goods divided into differently sized regions. The customer is forced to walk through the checkout on leaving the store to ensure that they pay for their goods. Listing B.3 in the appendix shows the full requirement specifications.

In order to force the customer to walk through the checkout when leaving this is the only room type connected to the entrance. Following the checkout every room type is allowed to connect to every other room type to allow a free arrangement of sections in which a customer can walk from any section to any other adjacent section.

One of the more interesting specifications is that the refrigerated sections of the store such as meats, frozen food, eggs and dairy require windows. They do not actually require windows, but this makes the system try to place these sections on the outside of the building which is most convenient for supplying refrigeration to from an outside utility. Additionally, these sections are to be within a short distance of each other to reduce the distance the cooling pipes will have to travel making it more efficient to construct.

Several sections have a minimum ratio specification as these are most often and conveniently laid out as one or two aisles with goods on either side. This minimum ratio rewards the system for creating long and narrow areas for these sections.

Figure 6.9 shows a selection of grocery stores evolved from the system. It has particular difficulty with this problem as one room of each type is required, making it difficult to combine solutions without violating this constraint. While missing several sections, the results are geometrically and logistically similar to the layout of a grocery store. This is certainly one class

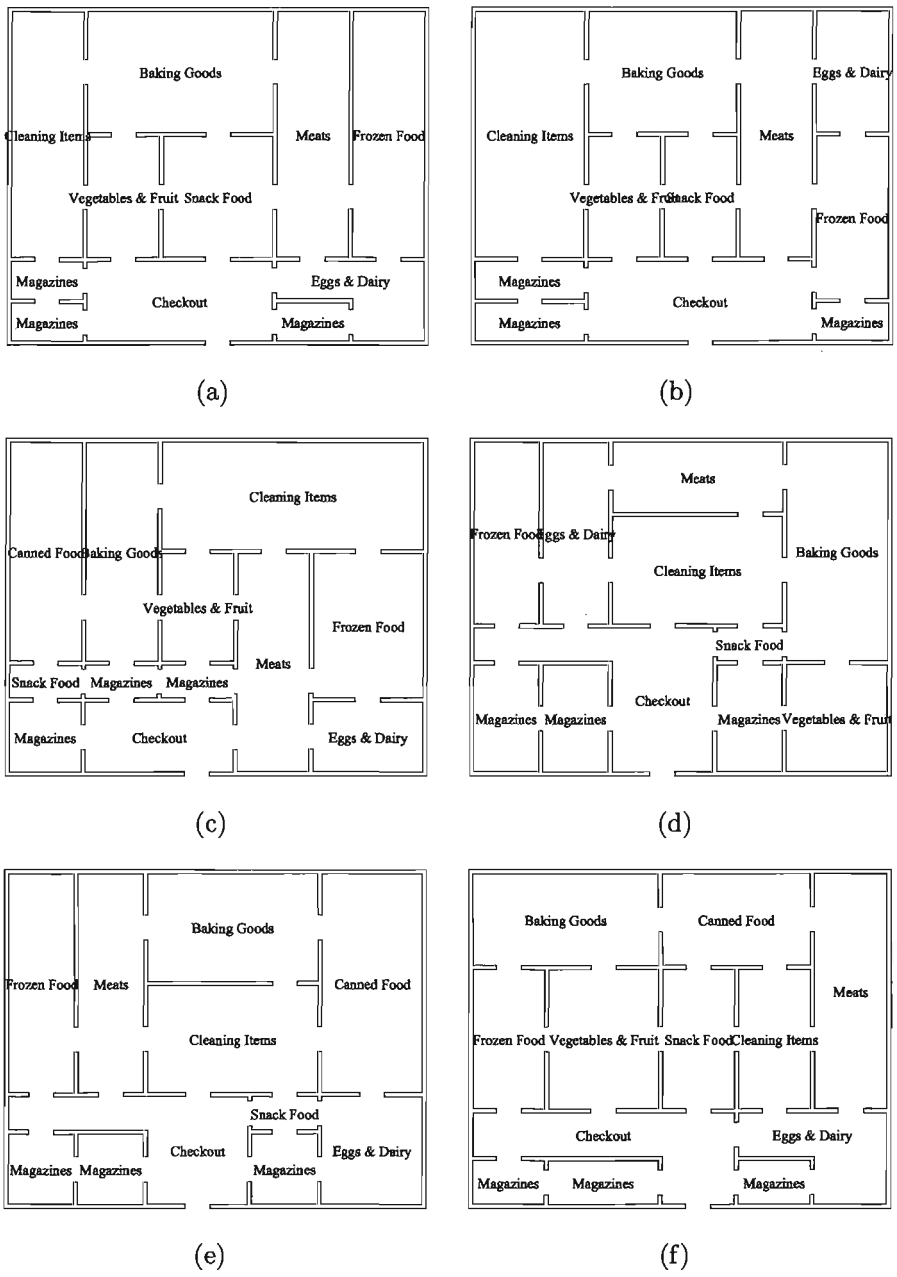


Figure 6.9: Grocery store layouts evolved by system

of problem requirements where the recombination strategy in this system is not as effective as one that would preserve room counts.

In all plans, the majority of requirements are fulfilled. Nevertheless, there remain a few issues in the solutions. Every store has three magazine sections. Stores (a) and (b) do not have a section for snack food. The optional flowers section is not in any of these layouts, hinting that there was some difficulty in satisfying the required rooms. Most of the layouts did manage to keep the cold foods together, though stores (e) and (f) separated eggs and dairy from the rest. In summary, the techniques in this thesis were easily adaptable to this unusual variation of problem requirements, with the production of interesting and often creative results.

Chapter 7

Discussion and Comparisons

7.1 Discussion

The system presented here works well to produce floor plans that not only satisfy its own fitness requirements, but are also as a result asthetically pleasing. The diversity preservation measure ensures that the resulting population contains a variety of plans so that the user will have different options to choose from.

As the description of what rooms are available and their requirements are loaded into the system via a simple configuration file, the system is easily adaptable to other problems with similar requirements. A few of these problems have been explored here-in but there are many other possible applications, even VLSI design.

Using evolutionary computation to solve this problem grants the ability to use easy solutions to complicated problems. As was shown with the case of polygonal rooms, a complete floor plan can be generated, a polygon cut out of it, and then evaluated for its quality afterwards. Whereas if the same naive strategy were applied to produce polygonal layouts from a procedural algorithm, the resulting plans would likely be unsatisfactory given that the procedural algorithm is not aware of what will be removed.

The procedural assignment algorithm greatly reduces the search space of the problem. By intelligently converting rooms into their appropriate types many useless layouts of houses are avoided. This allows the algorithm to work on much larger spaces or work much faster than with the assignment being part of what is evolved. The runs in this thesis took on average 7 seconds to

complete on a single core of an Intel Q6600 2.4Ghz processor. This means the algorithm is feasible to use in real-time applications. However, this assignment algorithm is also a limitation. Many configurations are impossible to evolve because the procedural algorithm will not assign them that way.

Scaling up to larger problems is challenging. The exponential growth in the search space gives the evolutionary system great difficulty in finding good solutions. This is a problem in any evolutionary computation search, which can often be leveraged by using modularization. By evolving modules such as bedroom areas, bath and utility areas, and living areas the search space could be greatly reduced. After these areas have been chosen small plans could be evolved within them.

The system also tends to cheat at various objectives. The multi-objective ranking methods help avoid cheating somewhat, but it is not uncommon for objectives to go unsatisfied as satisfying one objective can hurt the scores in several other objectives. In many cases the functional objective was not satisfied as having those rooms proved too difficult in terms of other objective scores like geometric and ratio.

Another weakness is in the evolution of hallway and other connecting type rooms. They are a special case of a rectilinear room and require a large portion of the chromosome to agree in order to create. This is one area which VLSI systems can often ignore as modules on a circuit do not need to be placed flush next to each other. Alternative means of constructing hallways could be examined in order to faciliate their construction.

7.2 Comparison

The ideal way to compare this system with previous work would be to run a test with the same objectives on both systems and see how their results compare. Unfortunately, most of the other work is either unrelated or does not have the same objectives. Table 7.1 gives an outline of the various features supported by similar works. Martin[21], Marson[19] and Mitchell[23] do not use a search and hence several features are not applicable in these works. These features are marked by dashes in the table.

The most relevant and similar work to what has been done here is by Doulgerakis[9]. This work has many similarities:

- This thesis uses an evolutionary system – namely a genetic program to construct the room layouts.

Table 7.1: Comparison of features in various works

System	Doulg[9]	Mart[21]	Marson[19]	Mitch[23]	Flack
GA construction					✓
GP construction	✓				✓
Procedural const.		✓	✓		
Optimal const.				✓	
Optimal assign.				✓	
Procedural assign.	✓	✓	✓		✓
Evolution assign.	✓				✓
Rectangular rooms	✓	✓	✓	✓	✓
Rectilinear rooms				✓	✓
Polygonal rooms	✓				
Polygonal exterior			✓		✓
Fixed rooms					✓
# of Objectives	4	–	–	1	6
Weighted Sum	✓	–	–	–	✓
Pareto Ranking		–	–	–	✓
Ranked Sum	✓	–	–	–	✓
Norm. Ranked Sum		–	–	–	✓

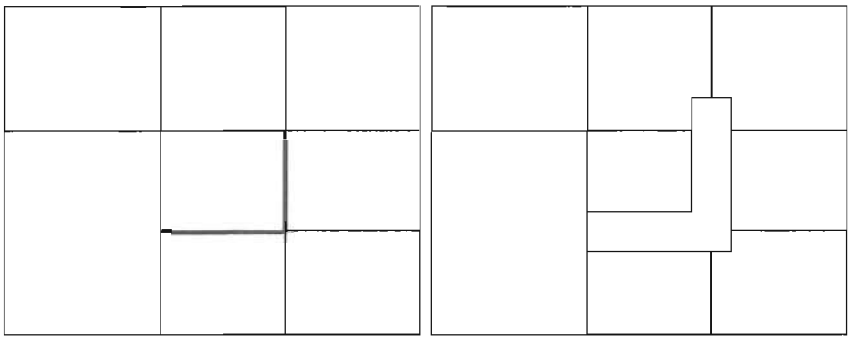
- The GP uses division of space in order to create the rooms from a program tree.
- Also uses a ranked-sum evaluation to balance multiple objectives as an alternative to weighted sum.
- Many of the objectives are the same as those used in this paper: ratio, size, number of, and connectivity of rooms.
- Doulgerakis also has two methods of assigning room types in the resulting floor plans; they can be assigned randomly as part of the chromosome much like the evolutionary room assignment described here, or they can be assigned using an "Assignment Embryology" similar to the procedural assignment used in this system.

Despite the great list of similarities, there are many differences in the work as well. In particular:

- This system has a genetic algorithm method of producing room layouts.
- Pareto ranking was implemented and compared to the ranked-sum multi-objective evaluation.
- Weights were added to the ranked sum in this system so that certain objectives could be prioritized.
- Some objectives were added, such as placing certain room types on the outside so that they can have windows and minimizing distances between related room types rather than just trying to connect them.
- Doulgerakis allows for angled splits of rooms whereas this system only allows for the creation of rectilinear rooms.

Ideally a direct comparison with a similar problem would be done, however the exact details of the problem Doulgerakis tackled were unfortunately not included in his thesis. Additionally there is not enough information given in the thesis to determine exactly how the fitness of the various objectives are measured.

Martin's[21] has a similar goal. He aims to construct floor plans which appear as though people would actually live in them. He does not worry



(a) Walls are selected to be converted into a hallway

(b) Selected walls are expanded becoming a hallway room.

Figure 7.1: Hallway conversion on rectangular floor plans

about fine tuning any aspects of the house, they are meant to be believable if observed as part of a level in a video game or the background of a large city in an animation. His system generates buildings using a procedural algorithm which constructs rooms following a table of statistics. It is rather similar to the procedural assignment phase of this system except that in addition to assigning room types it places the rooms and adjusts their sizes. This is designed with a very different goal in mind. There are no measured criteria of what is desirable or successful, and no choice in the resulting house.

Marson and Musse[19] developed a procedural algorithm for constructing floor plans using squarified treemaps. They too use a connectivity matrix to determine which room types will be connected in the resulting plan. A procedural algorithm runs after the house construction to ensure that rooms which are not connected will have a hallway joining them to the main living room. As with other procedural methods, the result is that you will either like the produced house or not, and it has not been subject to any user specified fitness measurements.

There is an interesting hallway construction strategy which can be seen in Figure 7.1. By expanding walls into hallways, complex room shapes do not need to be evolved allowing for a simpler smaller house representation. Instead, by using a procedural algorithm as in Marson's work, simple floor plans can be converted into much more desirable layouts quickly and easily. Alternatively the GA representation could be expanded to flag certain walls as hallways which would be later converted.

Mitchell *et al.*[23] were concerned with the evolution of floor plans that satisfied a certain set of adjacency requirements and areas. All possible floor plans were considered which as a result meant they could only work with very small search spaces. Unlike this thesis, there were no constraints such as increasing distance between certain rooms and all rooms were assumed to be connected. Additionally, the search for sizing of the rooms is done separately after determining good layouts whereas this thesis attempts to optimize the sizes and layout at the same time which could discover better plans.

Chapter 8

Conclusions

8.1 Results

The system in this thesis employed evolutionary algorithms to solve a problem with no clear strategy or notion of optimality. The system was effective at generating specific floor plans given reasonable requirements. The representation of the problem specifications allows it great flexibility in both problem depth and problem scope. Anything that can be defined in terms of rooms with dimensional and connectivity constraints can be evolved. Similarly, any level of detail in requirements may be specified. It works best on smaller problems, where it is capable of exploring a larger portion of the search space.

A number of insights into effective evolutionary computation strategies were found. While the genetic programming crossover was not conducive to the combination of good features from good solutions, the genetic algorithm succeeded in combining good features from good solutions. Several strategies for handling multiple objectives in evolutionary algorithms were tested and worked to evolve the often conflicting variety of goals in the design of floor plans. With its high dimensionality and unclear requirements, this problem served as an interesting benchmark problem to compare multi-objective strategies. While not conclusive, the normalized ranked sum strategy appeared to provide the best compromise of scores in order to find the best solutions. Lastly, diversity preservation was successfully employed to provide a more diverse search and a variety of end of run solutions to choose from.

The features provided in this work exceed the capabilities of any previous work done without a significant expansion to the search space. The polygonal floor plans and fixed room plans increase the utility of such a system such that it can be used for a more general set of problems. Additionally the variety of potential objectives and specification of requirements goes beyond that of similar works, to allow for greater flexibility in problem requirements.

8.2 Future Work

One major feature that would be useful for multi-floor structures is to evolve both floors as a single problem. This way the system will know that if there are many bedrooms on the second floor there do not need to be any on the first, or vice versa. In doing a multi-floor layout with the grid-based GA chromosome, the sizes of the grid cells could be constant for all floors with a separate array of values for each floor giving room types. This would help ensure structural integrity as walls would be built over walls rather than wherever they end up.

Considering it is largely unknown how to quantify which exact features make for a good house, it may be useful to apply an interactive/automatic hybrid evolutionary system. The user could pause the evolution, select houses that he/she believed were superior and those would be valued above the others. In this way the user would have a direct impact on the evolutionary process and the final resulting houses.

There are many parameters involved in the procedural assignment algorithm which have a great effect over how rooms are assigned. These values could be evolved as part of the GA chromosome to help find ideal values for assigning room types in a more desirable fashion. It would add a great deal of flexibility in room assignment and could relax the necessity for the user to give such precise requirements.

The fitness function is currently an evaluation of the floor plan based on several static measures. Since the true evaluation of the quality of a house is how well it serves the needs of its inhabitants, it may be possible to get a better rating of a house by running a simulation of agents using the house for various purposes. The evaluation may take significantly longer, however, it may be possible to find more relatable objectives with such a simulation.

There are also many other areas to which the process of evolving floor plans could be applied. Martin[20] uses his strategy in the application of

generating maps for role-playing games. It has been shown that the method described here has applications in other areas, however with some adaptation it may be better suited for these applications.

There are other possible representations for the chromosome in order to effectively evolve normal houses. For example, if the goal were to evolve strictly rectangular rooms there could be some rule for when to expand a cell or a width and height associated with each cell. Hallways could be more effectively constructed or evolved as edges between rooms as in "Automatic Real-time Generation of Floor Plans Based on Squarified Treemaps Algorithm"[19] which could be expanded to become hallways. This way the grid would not have to have cells specifically sized to allow hallways. Such modifications could effectively reduce the number of bad solutions in the search space which would render the algorithm effective for much larger spaces.

Similar to Marson's[19] algorithm, the GP representation could use a squarified treemap to lay out the rooms in the graph. This would allow for more effective crossover as entire trees of rooms would not be squished into small spaces as with the current strategy.

Bibliography

[1] Association for the advancement of artificial intelligence (aaai). Accessed August 2010 at `http://www.aaai.org/`.

[2] Google 3d warehouse. Accessed October 2010 at `http://sketchup.google.com/3dwarehouse/`.

[3] Google sketchup. Available online at `http://sketchup.google.com/`.

[4] Rafael Bidarra, Tim Tutenel, and M. Rubin Smelik. Rule-based layout solving and its application to procedural interior generation. 2009.

[5] M. Bruls, K. Huizing, and J. van Wijk. Squarified treemaps. In *Proc. of Joint Eurographics and IEEE TCVG Symp. on Visualization (TCVG 2000)*, pages 33–42. IEEE Press, 2000.

[6] David W. Corne and Joshua D. Knowles. Techniques for highly multiobjective optimisation: some nondominated points are better than others. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 773–780, New York, NY, USA, 2007. ACM.

[7] Charles Darwin. *The Origin of Species.* D. Appleton and Company, 1900.

[8] Richard C. Dorf, editor. *The electrical engineering handbook.* CRC Press, Inc., Boca Raton, FL, USA, 1993.

[9] Adam Doulgerakis. Genetic programming + unfolding embryology in automated layout planning. Master's thesis, University Of London, 2007.

[10] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Professional, 1 edition, January 1989.

[11] Stefan Greuter, Jeremy Parker, Nigel Stewart, and Geoff Leach. Real-time procedural generation of 'pseudo infinite' cities. In *Computer Graphics and Interactive Techniques*, page 8, Australasia and South East Asia, 2003.

[12] P. N. Gui, T. Takahashi, and C. K. Cheng. Floorplanning using a tree representation: A summary. In *Circuits and Systems Magazine*, volume 3, pages 26–29. IEEE Computer Society Press, November 2003.

[13] Evan Hahn, Prosenjit Bose, and Anthony Whitehead. Persistent realtime building interior generation. In *Sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, pages 179–186, New York, NY, USA, 2006. ACM.

[14] Bill. Hillier and Julienne. Hanson. *The social logic of space / Bill Hillier, Julienne Hanson.* Cambridge University Press, Cambridge [Cambridgeshire] ; New York :, 1984.

[15] William J. Hirsch Jr. *Designing Your Perfect House.* Dalsimer Press, 2008.

[16] John R. Koza. *Genetic programming: on the programming of computers by means of natural selection.* The MIT Press, Cambridge, MA, 1992.

[17] John R. Koza, Forrest, David Andre, and Martin A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, 1st edition, May 1999.

[18] Jens Lienig and James P. Cohoon. Genetic algorithms applied to the physical design of vlsi circuits: A survey. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, PPSN IV, pages 839–848, London, UK, 1996. Springer-Verlag.

[19] Fernando Marson and Soraia Raupp Musse. Automatic real-time generation of floor plans based on squarified treemaps algorithm. In *International Journal of Computer Games Technology.* Hindawi Publishing Corporation, 2010.

[20] Jess Martin. Generating graphical content using grammars and graphs. Technical report, Trinity University.

[21] Jess Martin. Algorithmic beauty of buildings: Methods for procedural building generation. Technical report, Trinity University, San Antonio, TX, USA, 2004.

[22] Melanie Mitchell. *An Introduction to Genetic Algorithms (Complex Adaptive Systems).* The MIT Press, February 1998.

[23] W J Mitchell, J P Steadman, and Robin S Liggett. Synthesis and optimization of small rectangular floor plans. *Environment and Planning B: Planning and Design*, 3(1):37–70, 1976.

[24] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. *ACM Trans. Graph.*, 25(3):614–623, 2006.

[25] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming.* Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[26] Jan Prideaux and Marian E. Haggard, editors. *Contemporary Home Plans: 220 Sleek Designs for Modern Lifestyles.* Home Planners, 1998.

[27] Stuart J. Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, second edition, 2003.

[28] Majid Sarrafzadeh. Transforming an arbitrary floorplan into a sliceable one. In *ICCAD '93: Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 386–389, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[29] Thorsten Schnier and John S. Gero. Learning genetic representations as alternative to hand-coded shape grammars. In *Gero & F. Sudweeks (eds), Artificial Intelligence in Design '96, Kluwer*, pages 39–57, 1996.

[30] Suphachai Sutanthavibul, Eugene Shragowitz, and J. Ben Rosen. An analytical approach to floorplan design and optimization. In *DAC '90: Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 187–192, New York, NY, USA, 1990. ACM.

[31] Maolin Tang and Alvin Sebastian. A genetic algorithm for vlsi floorplanning using o-tree representation. In Franz Rothlauf, Jürgen Branke, Stefano Cagnoni, David W. Corne, Rolf Drechsler, Yaochu Jin, Penousal Machado, Elena Marchiori, Juan Romero, George D. Smith, and Giovanni Squillero, editors, *Applications on Evolutionary Computing*, volume 3449 of *Lecture Notes in Computer Science*, pages 215–224. Springer Berlin / Heidelberg, 2005.

[32] Christine L. Valenzuela and Pearl Y. Wang. A genetic algorithm for vlsi floorplanning. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, PPSN VI, pages 671–680, London, UK, 2000. Springer-Verlag.

[33] D. F. Wong and C. L. Liu. A new algorithm for floorplan design. In *Proc. 24th Design Automation Conference*, pages 101–107, June 1986.

Appendix A

Penalties for procedural assignment

The penalties for the procedural assignment algorithm are outlined below. The score of a room is the sum of the penalties and the room type with the lowest overall value is chosen first.

Penalty	Value
The new room's area is larger than another room, that it should be smaller than.	2000
The room has enough twists to be recommended to be a particular type.	1000
The room type recommends a private/public room and the opposite type is being considered.	700
The room is an external one and a type that does not need windows is being considered.	150

The room's area, *area*, is less than the minimum recommended area for this type, *minarea*.	$500 + minarea - area$
The room's area, *area*, is greater than the maximum recommended area for this type, *maxarea*.	$500 + area - maxarea$
The room's width, *width*, is less than the minimum recommended width for this type, *minwidth*.	$500 + (minwidth - width)^2$
The room's width, *width*, is greater than the maximum recommended width for this type, *maxwidth*.	$500 + (maxwidth - width)^2$
The room with width *width*, has a recommended width, *recwidth*.	$(width - recwidth)^2$
The room with area *area*, has a recommended area, *recarea*.	$\|area - recarea\|$
The room with ratio *ratio*, has a smaller ratio than the recommended minimum ratio, *minratio*.	$200 \cdot (minratio - ratio)$
The room with ratio *ratio*, has a greater ratio than the recommended maximum ratio, *maxratio*.	$200 \cdot (maxratio - ratio)$
If the house does not have enough of the current type.	-125

If there are n too many of the current room type in the house.	$200n$

Appendix B

Requirements for experiments in thesis

Listing B.1: Requirements for "real" house

```
Outside
{
  type: public;
  attach: Entry;
}

Living
{
  type: public;
  minimum: 1;
  maximum: 1;
  attach: Dining, Eating, Hallway;
  min-width: 12;
  min-area: 175;
  max-ratio: 1.5;
  bigger-than: Bedroom, Kitchen, Dining, Eating, Bathroom, M
     Bathroom, M Bedroom, Entry;
}

Dining
{
  type: public;
  minimum: 1;
  maximum: 1;
  attach: Living, Kitchen;
  min-width: 10;
```

```
  min-area: 100;
  max-ratio: 1.5;
  bigger-than: Bedroom;

  access-nearest: Bathroom;
  area: 120;
}

Bedroom
{
  type: private;
  minimum: 1;
  maximum: 2;
  min-width: 10;
  min-area: 100;
  max-ratio: 1.5;
  bigger-than: Bathroom, M Bathroom;

  access-nearest: Bathroom;
  windows: yes;
}

Hallway
{
  type: public;
  maximum: 1;
  attach: Bathroom, Bedroom, M Bedroom;
  width: 5;
  min-ratio: 2;
}

Kitchen
{
  type: public;
  min-width: 11;
  min-area: 100;
  minimum: 1;
  maximum: 1;
  max-ratio: 1.5;
  attach: Eating, Dining;
  bigger-than: Bathroom;
}

Bathroom
{
```

```
  type: private;
  minimum: 1;
  maximum: 2;
  max-ratio: 1.5;
  min-width: 5;
  min-area: 35;
  windows: yes;
}

M Bathroom
{
  type: private;
  minimum: 1;
  maximum: 1;
  max-ratio: 1.5;
  min-width: 5;
  min-area: 35;
  bigger-than: Bathroom;
  windows: yes;
}

M Bedroom
{
  type: private;
  minimum: 1;
  maximum: 1;
  min-width: 10;
  min-area: 100;
  max-ratio: 1.5;
  attach: M Bathroom;
  bigger-than: Bedroom, M Bathroom, Bathroom;
  windows: yes;
}

Eating
{
  type: public;
  minimum: 1;
  maximum: 1;
  attach: Living, Kitchen;
  min-width: 10;
  min-area: 100;
  max-ratio: 1.5;
  bigger-than: Bedroom;
```

```
  access-nearest: Bathroom;
  area: 120;
}

Entry
{
  type: public;
  maximum: 1;
  attach: Living;
  max-width: 8;
}
```

Listing B.2: Requirements for office building

```
Outside
{
  type: public;
  attach: Hallway;
}

Lab
{
  type: public;
  minimum: 2;
  maximum: 2;
  min-width: 7;
  min-area: 150;
  max-ratio: 2;
  bigger-than: Office, Bathroom;
}

Bathroom {
  type: private;
  minimum: 1;
  maximum: 2;
  min-width: 5;
  min-area: 35;
  max-ratio: 2;
  bigger-than: Closet;

  width: 6;
  area: 54;
}

Office {
  type: private;
```

```
  minimum: 12;
  windows: yes;
  min-width: 7;
  min-area: 80;
  max-ratio: 2;
  bigger-than: Bathroom;

  access-nearest: Bathroom;
  area: 150;
}

Hallway {
  type: public;
  minimum: 1;
  min-ratio: 2;
  width: 4.5;
  attach: Office, Lab, Closet, Bathroom;
  twists: 3;
}

Closet{
  type: private;
  minimum: 0;
  max-ratio: 3;
}
```

Listing B.3: Requirements for grocery store

```
Outside
{
  type: public;
  attach: Checkout;
}

Checkout
{
  minimum: 1;
  maximum: 1;
  min-ratio: 3;
  area: 240;
  max-area: 300;
  min-area: 150;
  rectangular: 0.9;
  attach: Magazines, Frozen Food, Eggs & Dairy, Meats,
      Vegetables & Fruit, Cleaning Items, Baking Goods, Canned
      Food, Snack Food, Flowers;
```

```
  access: Snack Food, Magazines;
}

Magazines
{
  minimum: 1;
  maximum: 1;
  area: 80;
  max-area: 120;
  attach: Magazines, Frozen Food, Eggs & Dairy, Meats,
      Vegetables & Fruit, Cleaning Items, Baking Goods, Canned
      Food, Snack Food, Flowers;
}

Frozen Food
{
  minimum: 1;
  maximum: 1;
  min-ratio: 2;
  windows: yes;
  area: 220;
  min-area: 150;
  bigger-than: Magazines, Eggs & Dairy, Snack Food, Flowers;
  attach: Magazines, Frozen Food, Eggs & Dairy, Meats,
      Vegetables & Fruit, Cleaning Items, Baking Goods, Canned
      Food, Snack Food, Flowers;
  access: Meats;
}

Eggs & Dairy
{
  minimum: 1;
  maximum: 1;
  windows: yes;
  area: 80;
  min-area: 60;
  attach: Magazines, Frozen Food, Eggs & Dairy, Meats,
      Vegetables & Fruit, Cleaning Items, Baking Goods, Canned
      Food, Snack Food, Flowers;
  access: Frozen Food, Meats;
  bigger-than: Flowers, Magazines;
}

Meats
{
```

```
  minimum: 1;
  maximum: 1;
  min-ratio: 3;
  windows: yes;
  area: 240;
  min-area: 170;
  access: Eggs & Dairy;
  attach: Magazines, Frozen Food, Eggs & Dairy, Meats,
      Vegetables & Fruit, Cleaning Items, Baking Goods, Canned
      Food, Snack Food, Flowers;
  bigger-than: Magazines, Eggs & Dairy, Flowers;
}

Cleaning Items
{
  minimum: 1;
  maximum: 1;
  area: 240;
  min-ratio: 2;
  attach: Magazines, Frozen Food, Eggs & Dairy, Meats,
      Vegetables & Fruit, Cleaning Items, Baking Goods, Canned
      Food, Snack Food, Flowers;
}

Baking Goods{
  minimum: 1;
  maximum: 1;
  area: 240;
  min-ratio: 2;
  bigger-than: Frozen Food, Eggs & Dairy, Meats;
  attach: Magazines, Frozen Food, Eggs & Dairy, Meats,
      Vegetables & Fruit, Cleaning Items, Baking Goods, Canned
      Food, Snack Food, Flowers;
}

Canned Food{
  minimum: 1;
  maximum: 1;
  min-ratio: 2;
  area: 240;
  attach: Magazines, Frozen Food, Eggs & Dairy, Meats,
      Vegetables & Fruit, Cleaning Items, Baking Goods, Canned
      Food, Snack Food, Flowers;
}
```

```
Snack Food{
  minimum: 1;
  maximum: 1;
  area: 100;
  attach: Magazines, Frozen Food, Eggs & Dairy, Meats,
      Vegetables & Fruit, Cleaning Items, Baking Goods, Canned
      Food, Snack Food, Flowers;
}

Flowers{
  maximum: 1;
  area: 180;
  attach: Magazines, Frozen Food, Eggs & Dairy, Meats,
      Vegetables & Fruit, Cleaning Items, Baking Goods, Canned
      Food, Snack Food, Flowers;
}

Vegetables & Fruit
{
  minimum: 1;
  maximum: 1;
  area: 140;
  attach: Magazines, Frozen Food, Eggs & Dairy, Meats,
      Vegetables & Fruit, Cleaning Items, Baking Goods, Canned
      Food, Snack Food, Flowers;
}
```

Appendix C

Full statistical data

Table C.1: Confidence in Directed Search over Random Search

Average Best Fitness						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
Random	8.0	260.3	0.1667	2.034	4.768	0.03333
Directed	7.0	72.06	0.0	1.904	1.063	0.0

Standard Deviation						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
Random	3.256	400.7	0.3727	0.1503	3.139	0.1795
Directed	0.0	155.4	0.0	0.08078	1.56	0.0
Confidence	**95.4%**	**99.2%**	**99.3%**	**99.9%**	**99.9%**	84.6%

Table C.2: Confidence in Genetic Algorithm over Genetic Programming approach

Average best fitness						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
GA	7.0	72.06	0.0	1.904	1.063	0.0
GP	9.333	245.2	0.4667	1.866	2.299	0.0

Standard Deviations						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
GA	0.0	155.4	0.0	0.08078	1.56	0.0
GP	3.902	345.6	0.6182	0.104	3.542	0.0
Confidence	**99.9%**	**99.4%**	**99.9%**	5.585%	**96.0%**	50.0%

Table C.3: Confidence in Procedural assignment outperforming Evolutionary

Average best fitness						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
Procedural	7.0	72.06	0.0	1.904	1.063	0.0
Evolutionary	7.417	3.584	0.0	54.43	0.6798	0.0

Standard Deviations						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
Procedural	0.0	155.4	0.0	0.08078	1.56	0.0
Evolutionary	5.941	13.08	0.0	86.38	1.499	0.0
Confidence	65.0%	0.80%	50.0%	**99.9%**	16.6%	50.0%

Table C.4: Statistical comparison of multi-objective ranking strategies
D = Diversity preservation, N = Normalized

Average best fitness						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
Weighted	7.333	0.0	0.1667	1.827	0.0020	0.0
Pareto	7.333	13.97	0.0333	1.9	1.462	0.0
Ranked	7.0	0.0	0.0	1.875	0.2073	0.0
N. Rank.	7.0	0.0	0.0	1.879	0.0	0.0
D. Weighted	7.533	0.0	0.1333	1.789	0.2233	0.0
D. Pareto	7.0	72.06	0.0	1.904	1.063	0.0
D. Ranked	7.0	0.0	0.0	1.882	0.2016	0.0
D. N. Ranked	7.0	0.0	0.0	1.877	0.0	0.0

Standard Deviations						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
Weighted	0.7454	0.0	0.3727	0.1046	0.0109	0.0
Pareto	1.795	57.25	0.1795	0.0797	2.16	0.0
Ranked	0.0	0.0	0.0	0.0025	0.778	0.0
N. Rank.	0.0	0.0	0.0	0.0224	0.0	0.0
D. Weighted	0.8844	0.0	0.3399	0.0789	0.75	0.0
D. Pareto	0.0	155.4	0.0	0.0808	1.56	0.0
D. Ranked	0.0	0.0	0.0	0.0226	0.6516	0.0
D. N. Ranked	0.0	0.0	0.0	0.0055	0.0	0.0

Confidence in Diverse Normalized Ranked over other methods:						
Weighted	**99.3%**	50.0%	**99.3%**	0.48%	84.6%	50.0%
Pareto	84.6%	90.9%	84.6%	94.2%	**99.9%**	50.0%
Ranked	50.0%	50.0%	50.0%	12.1%	92.8%	50.0%
N. Rank.	50.0%	50.0%	50.0%	71.6%	50.0%	50.0%
D. Weighted	**99.9%**	50.0%	**98.4%**	0.01%	94.9%	50.0%
D. Pareto	50.0%	**99.5%**	50.0%	**96.7%**	**99.9%**	50.0%
D. Ranked	50.0%	50.0%	50.0%	88.9%	**95.5%**	50.0%

Table C.5: Confidence in weighting objective values over equal weighting

Average best fitness						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
Equal	7.833	0.0	3.233	1.274	0.0	0.0
Weighted	7.0	72.06	0.0	1.904	1.063	0.0

Standard Deviations						
Objective	Connect.	Geom.	Funct.	Reach.	Ratio	Windows
Equal	2.083	0.0	2.093	0.5043	0.0	0.0
Weighted	0.0	155.4	0.0	0.08078	1.56	0.0
Confidence	**98.6%**	0.55%	**99.9%**	0.01%	0.01%	50.0%

Appendix D

Detailed fitness penalties

Listing D.1: Complete description of penalties to reconstructed real house

```
GEOMETRIC: M Bedroom has width 6.44444, less than the desired
    10, penalty of 12.642
CONNECTIVITY: Distance from Dining to nearest Bathroom is 3
    scaled to 3
CONNECTIVITY: Distance from Bedroom to nearest Bathroom is 3
    scaled to 1.5
CONNECTIVITY: Distance from Bedroom to nearest Bathroom is 3
    scaled to 1.5
CONNECTIVITY: Distance from Eating to nearest Bathroom is 3
    scaled to 3
REACHABLE: Depth of Outside is 0, scaled to 0
REACHABLE: Depth of Kitchen is 4, scaled to 0.333333
REACHABLE: Depth of Dining is 3, scaled to 0.25
REACHABLE: Depth of Entry is 1, scaled to 0.0833333
REACHABLE: Depth of Bedroom is 4, scaled to 0.333333
REACHABLE: Depth of Bedroom is 4, scaled to 0.333333
REACHABLE: Depth of Eating is 3, scaled to 0.25
REACHABLE: Depth of Living is 2, scaled to 0.166667
REACHABLE: Depth of Hallway is 3, scaled to 0.25
REACHABLE: Depth of M Bedroom is 4, scaled to 0.333333
REACHABLE: Depth of Bathroom is 4, scaled to 0.333333
REACHABLE: Depth of M Bathroom is 5, scaled to 0.416667
RATIO: Bathroom has larger ratio than 1.5, its ratio is 1.99049,
     penalty of 0.490488
RATIO: M Bathroom has larger ratio than 1.5, its ratio is
    1.99049, penalty of 0.490488
```